(2)

THE ANALYTIC SCIENCES CORPORATION

TR-9033-3

TASK ORDER 33 SOFTWARE MEASUREMENT PROCESS

SDS Software Measurement Plan
Technical Report

14 July 1989

Prepared for:

STRATEGIC DEFENSE INITIATIVE ORGANIZATION
Office of the Secretary of Defense
Washington, D.C. 20301-7100

Prepared under:

Contract No. SDIO84-88-C-0018

Prepared By:

SPARTA, Inc.
Teledyne Brown Engineering
The Analytic Sciences Corporation

THE ANALYTIC SCIENCES CORPORATION
1700 North Moore Street
Suite 1800
Arlington, VA 22209

TASC TBE SPARTA ARI GRC DSA
THE INTEGRATED SETA TEAM COMMITTED TO SDIO

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-9033-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Analytic Sciences Corporation (Prime Contractor) 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Strategic Defense Initiative Organization
6c. ADDRESS *(City, State, and ZIP Code)* 1700 N. Moore Street Suite 1800 Arlington, VA 22209	7b. ADDRESS *(City, State, and ZIP Code)* Room 1E149 The Pentagon Washington, D.C. 20301-7100
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Organization 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* Room 1E149 The Pentagon Washington, D.C. 20301-7100	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*

SDS Software Measurement Plan (U)

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM 5Dec88 TO 14Jul89	1989, July, 14	67

16. SUPPLEMENTARY NOTATION TASC Report No. TR-9033-3
Prepared by SPARTA, Inc, Teledyne Brown Engineering, and The Analytic Sciences Corporation

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
SDS SOFTWARE, EVALUATION, MEASUREMENT PROCESS, METRICS, MEASUREMENT PLAN

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report presents a plan for the introduction of software measurement into the SDS development program. This plan was developed based on an evaluation of SDS Software measurement requirements, and an assessment on software measurement processes and availability of measurement support tools. The plan focuses on both near term and longer term tasks necessary to initiate and maintain a viable software measurement program for the SDS. Near term tasks are oriented toward review, assimilation, and refinement of a proposed SDS Software Measurement approach, and use of existing software metrics tools. Longer term tasks target modeling of software development life cycles and definition of additional metrics development, validation, and automated implementation.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL [illegible]	22b. TELEPHONE *(Include Area Code)* (202) 693-1600	22c. OFFICE SYMBOL SDIO/S/ENA

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

89 9 20 098

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

PURPOSE AND SCOPE

The work presented in this report was performed under Subtask 4 of Task Order 33 of the SDIO Systems SETA contract. The purpose of the subtask was to develop a plan for the introduction of software measurement into the SDS development program. Sparta was the technical lead on the subtask, with support from Teledyne Brown and TASC.

The subtask was broken into two phases. The first phase focused on a review of the results of Subtasks 1, 2, and 3, which presented SDS Software Measurement requirement, models, and tools, and near term efforts to advance SDS Software Measurement plans. The second phase focused on the definition of activities that provide for a stronger foundation for the program in the longer term, with an emphasis on model development.

REVIEW OF REQUIREMENTS, METRICS, TOOLS

The primary purpose of software metrics is to predict, throughout the software development phase, what the quality and overall schedule and cost of the final product will be. This prediction process is to be integrated with the standard review and audit activities of existing software development methodologies for the SDS. As a basis for quality prediction, nine quality factors were defined. Lines of code, or function points, were the primary basis for productivity predictions. The SDS software was decomposed into 36 different software application domains, in order to produce more homogeneous groupings for prediction purposes. A framework methodology requirement was defined, consisting of specification, estimation, evaluation, and tuning phases.

The collection of software metrics available for use proved, as was expected, to be concentrated late in the software development life cycle, often with no formal foundations. Productivity models and supporting tools are more mature than quality-oriented models and tools; COCOMO and its variations are in widespread use. While not being oriented toward prediction, the software quality and management indicators standardized by the Army and Air Force can assist in providing early and consistent software productivity and quality statistics. A movement toward a multi-attribute vector representation of quality factors is identified. Specific tools packages which are applicable to SDS needs are "SMART" (Software Management and Reporting Tool) and "SMERFS" (Statistical Modeling and Estimation of Reliability Functions for Software).

NEAR TERM ACTIVITIES

To support the introduction of software measurement into SDS software development, a review of the conclusions and proposals presented in TR-9033-1 and TR-9033-2 is proposed. A series of briefings, with audience feedback, is defined to present the SDS Software Measurement materials to major SDS contractors and associated Government personnel. The intent is both to educate the audience on SDIO plans in this area, and to profit from their evaluations of the SDS Software Measurement approach, based on their unique experiences and knowledge.

Based on the set of candidate software metrics models and tools previously identified, an initial measurement system should be developed for SDS. The system would be used to perform initial experiments in software characteristics prediction, with a small number of software development efforts. Cost, schedule, and reliability predictions would the the focus of initial experiments with the system.

LONGER TERM ACTIVITIES

Using the initial SDS Software Measurement System for experiments to gain experience is a necessary precursor to future SDS Software Measurement activities. From the results of the experiments, and the already recognized software metrics areas of deficiencies, requirements would be identified for the next phase of development. These requirements would include models for reliability, integrity, portability, usability, and reusability. One promising area of development appears to be in multi-variate vector metrics, as it has been successfully applied to the field of general systems optimization modelling.

In addition, the measurement methodology itself would be more completely defined. This definition would include definition of SDS software development methodology to include prototyping methodology. The software measurement methodology would be fully integrated with the expanded development methodology. The redefinition would also include further development of the software development methodology to incorporate methodologies for rating quality, model tuning, and metrics selection. Finally, an integrated system of automated tools would be developed to support the entire suite of SDS Software Measurement functions.

0. INTRODUCTION

This document presents the tasks to be accomplished in order to define a program of software measurement for SDS development, and to introduce that program into the SDS program. The plan focuses on both near term and longer term tasks necessary to initiate and maintain a viable software measurement program for the SDS. Near term tasks are oriented toward review, assimilation, and refinement of a proposed SDS Software Measurement approach, and use of existing software metrics tools. Longer term tasks target modelling of software development life cycles and definition of additional metrics development, validation, and automated implementation.

Section 1 discusses the importance of software measurement in supporting SDS software development. Section 2 reviews the findings of TR-9033-1 (SDS Software Measurement Requirements) and TR-9033-2 (Evaluation of Software Measurement Processes and Software Measurement Support). It also lists near-term activities to be undertaken to move the SDS Software Measurement program forward. Section 3 presents longer term activities, including additional software metrics validation and integration, and development of automated tools and databases. Appendix A presents a multi-metric measurement model with a formal mathematical foundation.

1. SDS SOFTWARE MEASUREMENT GOALS/OBJECTIVES

Software measurement is recognized as an important contributor to the successful development of the Strategic Defense System (SDS). While the exact extent of the software necessary to define, simulate, prototype, implement, test, deploy, and maintain the SDS is not known, a common indicator of the scope of the effort has been the estimate of between 10 and 90 million lines of code. This is a daunting number, even to those who previously developed large systems.

Size is not the only challenging aspect of SDS software development. Necessarily distributed across many years, it also will likely be accomplished by several dozens of individual contractors and subcontractors. In addition, the SDS encompasses a broad collection of differing functions, ranging from mundane (accounting, formatting tools, etc) to esoteric (nuclear detonation detection/analysis, interceptor guidance, etc.). Some needs will be met by off-the-shelf software products; others will require extended development. During the period of development, neither development technology, system design, nor hardware base can be expected to remain static. New technology must be planned for and accommodated within the SDS development plans.

One other characteristic of the SDS has a major impact on software development. SDS mission critical software must perform as designed. The consequences of SDS software failing to perform, or performing correctly, but under the incorrect circumstances, could be catastrophic. Thus, the criticality of the mission requires the most reliable, correct software we can build. The costs of errors in that software are grouped in two categories, then: (1) cost of failed or incorrect mission accomplishment, and (2) cost of preventing/identifying/removing errors in the development process. Software measurement can be used to assist in minimizing the costs associated with (2) above, while vigorous testing and evaluation addresses overall cost of error minimization.

SDS development is expected to be phased over many years, with hundreds or thousands of interrelated tasks to be accomplished. Software development must be accomplished in accordance with the overall schedule. Without firm monitoring and controls on progress, software development has the potential for delaying SDS completion by perhaps years. In addition, the costs associated with these delays could escalate to prohibitive levels.

SDS software measurement, then, has as its goal to: (a) measure the product, and process, of SDS software development, both to predict results, and to allow feedback and control, and (b) to achieve a superior product, on time and within budget. The key concept here is prediction. With accurate prediction early in the software development life cycle, discrepancies between expected and required results can be identified, and appropriate steps taken. Without the prediction afforded by software measurement, problems will remain hidden longer, requiring additional resources to resolve. Thus, software measurement is the "early warning system" in the management of SDS software development.

2. SDS SOFTWARE MEASUREMENT NEAR-TERM APPROACH

This sectior discusses the near-term activities to be undertaken to introduce in a systematic manner the software measurement process into SDS Software Development. It summarizes the findings documented in TR-9033-1 and TR-9033-2, and proposes a method for review of, and concurrence with those findings, by Government and other contractor personnel. This section also defines an incremental approach to software measurement introduction, using an initial tailored software metrics program with one or more development contracts. It also includes a recommendation for experiments.

2.1 SUMMARY OF TR-9033-1 AND TR-9033-2

This section summarizes the findings from Subtasks 1, 2, and 3 concerning measurement requirements by software application domains, and the models, metrics, tools, and environments that support those measurements.

2.1.1 Summary of SDS Software Measurement Requirements (TR-9033-1)

The work presented in this report was performed under Subtask 1 of Task Order 33 of the SDIO Systems SETA contract. The purpose of the subtask was to develop SDS software measurement requirements by identifying the standards and attributes required for SDS software products and the software development process.

The subtask was broken into three phases. The first phase was a review of standards and development process relevant to SDS, as they pertained to software metrics. The second phase was a detailed examination of SDS software characteristics, definition of software domains, and quality and productivity measurement requirements. The third phase integrated the measurement requirements identified previously into an approach for the application of software metrics.

Standards and Process Identification – For purposes of this document, software measurement is defined as the act of capturing metrics and comparing them to standards. A metric is defined as a quantitative standard of measurement used to represent and compare some software process or product attribute. The primary objective of software metrics is to predict, throughout the development phase of the software, what the quality and overall schedule and cost of the final product will be. The task began with the identification of potentially relevant standards and development guidelines. Many were found to be of too high a level for specific guidance, or had no relevance to software metrics technology. The ones which were found to most directly address metric requirements were:

a) SDS Software Policy and Management Directive No. 7;

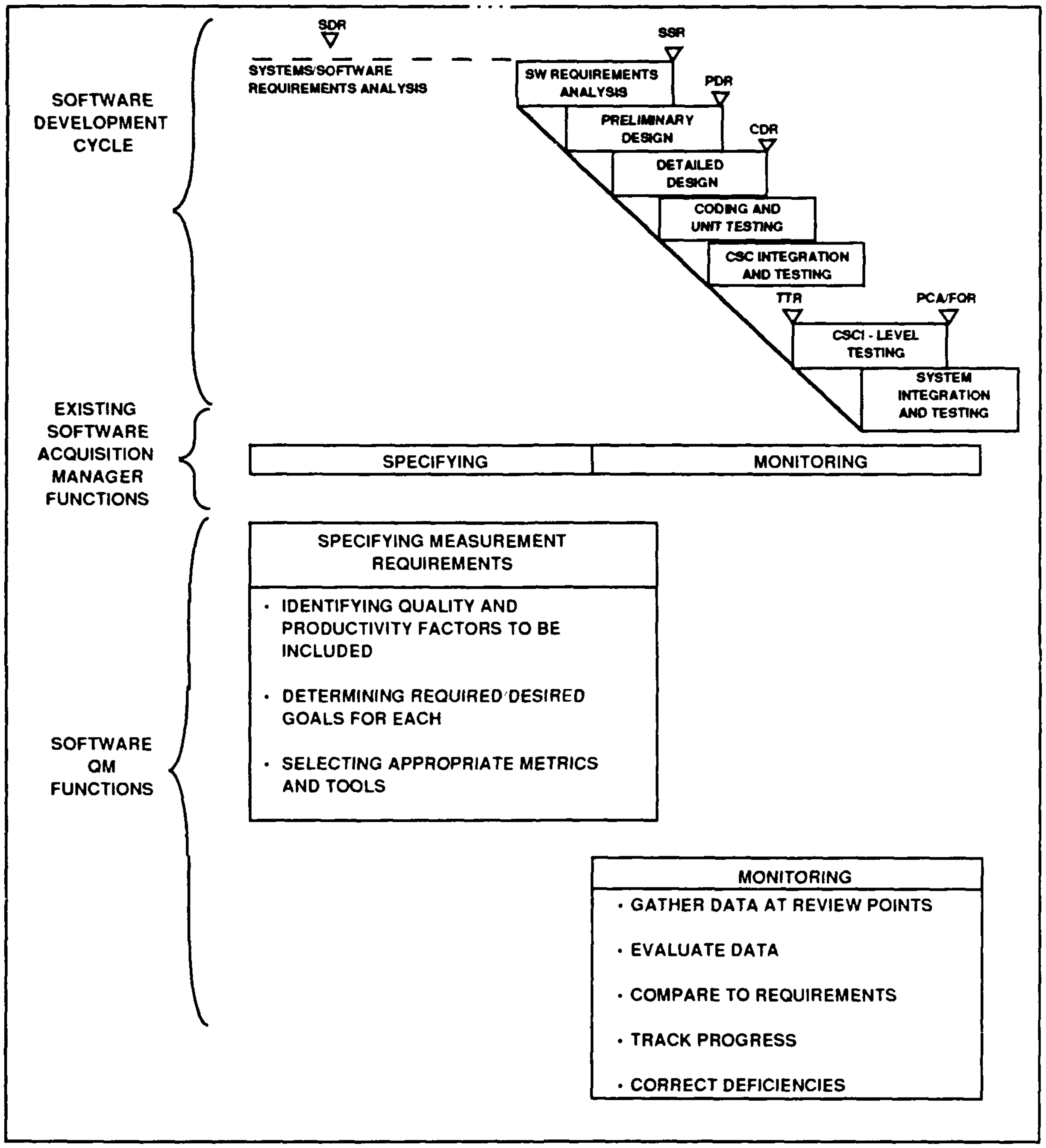
b) DoD-STD-2167A, Defense Systems Software Development;

c) RADC-TR-85-37, Vols. 1, 2, and 3, Specification of the Software Quality Attributes.

These documents formed the core of the analysis, as they most directly addressed the development process and the associated software attributes. Integration of software metrics into the software development process was examined, including both the waterfall development method and the rapid prototyping development method. While metrics integration with the waterfall development method was relatively straightforward to define (see Figure 2.1-1, Software Acquisition Quality Metric Functions), integration with the rapid prototyping development method (itself not well understood) was not as complete.

Products and Process Attributes – The second phase of this subtask was to define the software quality and productivity factors relevant to SDS software. This identification spanned the software development life cycle from requirement definition through operation and maintenance. For the quality factors, the thirteen factors identified by RADC were examined and ·:· lyzed for applicability to SDS needs. Several were redefined or merged together, to give a new set of nine quality factors. Productivity factors were also examined, with factors relating to schedule, budget, and feedback for improvement. Primary sources for analysis were based on lines of code estimates for software components, or the identification of function points. Both quality and productivity metrics are believed to be much more accurate when applied against subsets of "like" software (software domains). Accordingly, the SDS software functions were categorized and decomposed into 36 SDS software application domains having disti

TASC THE SPARTA ARI GRC DSA
THE INTEGRATED SETA TEAM COMMITTED TO SDIO


SOFTWARE DEVELOPMENT CYCLE
SDR
SYSTEMS/SOFTWARE REQUIREMENTS ANALYSIS
SSR
SW REQUIREMENTS ANALYSIS
PDR
PRELIMINARY DESIGN
CDR
DETAILED DESIGN
CODING AND UNIT TESTING
CSC INTEGRATION AND TESTING
TTR
PCA/FQR
CSCI - LEVEL TESTING
SYSTEM INTEGRATION AND TESTING
EXISTING SOFTWARE ACQUISITION MANAGER FUNCTIONS
SPECIFYING
MONITORING
SOFTWARE QM FUNCTIONS
SPECIFYING MEASUREMENT REQUIREMENTS
• IDENTIFYING QUALITY AND PRODUCTIVITY FACTORS TO BE INCLUDED
• DETERMINING REQUIRED/DESIRED GOALS FOR EACH
• SELECTING APPROPRIATE METRICS AND TOOLS
MONITORING
• GATHER DATA AT REVIEW POINTS
• EVALUATE DATA
• COMPARE TO REQUIREMENTS
• TRACK PROGRESS
• CORRECT DEFICIENCIES

0389/002-002

Figure 2.1-1 Software Acquisition Quality Metric Functions

measurement requirements (Figure 2.1-2). Each domain was described through the function implemented, its level of criticality, time constraints, location, size, risk, use, and intended life cycle, as well as its quality attributes. This division served then as the basis for further measurement requirements definition.

Methodology Requirements – The third phase of this subtask was to define methodology requirements for incorporation of software measurement into the software development methodology. The approach defined was: specification, estimation, evaluation, and tuning. The specification phase is perhaps the most difficult, requiring negotiation among developers, users, and contracting agencies to set specific requirements. The estimation phase uses the appropriate metrics to obtain predicted measures of product and process. Upon delivery, the user evaluates the actual product, and that evaluation is compared to the metrics predictions. Finally, the metrics are adjusted as necessary to provide more accurate assessments for future developments.

Conclusions – There were several major conclusions from this subtask. The results of the review of available standards reveaied that formalization of the process of estimating quality by embedding it in the development process is not well-defined. Rapid prototyping, in particular, must be directly addressed. For the waterfall development, the incorporation process was identified. A modified set of quality factors tailored to SDS requirements was proposed, reducing metric application requirements while targeting specific needs. We developed a methodology for setting quality factors based on SDS software characteristics. The SDS software was decomposed into 36 software application domains for purposes of measurement application. The concept of software level to determine the extent of software metrics application was described. Several promising productivity metrics, were identified, which appear applicable to the SDS software domains. A four phase methodology requirement of software metrics application was also developed.

QUALITY ATTRIBUTE RELATIVE RANKING / FUNCTION		RELIABILITY	SURVIVABILITY	INTEGRITY	THROUGHPUT	USABILITY	MAINTAINABILITY	PORTABILITY	REUSE	EFFICIENCY
2.4.1	DETECT PLUMES	H	H	H	H	L	L	L	L	M
2.4.2	DETECT COLD BODIES	H	H	H	H	L	L	L	L	H
2.4.3	RF DETECT	H	H	H	H	L	M	M	L	M
2.4.4	RESOLVE OBJECTS	H	H	H	M	L	L	L	L	M
2.4.5	DISCRIMINATE	H	H	H	M	L	L	L	L	M
2.4.6	ASSESS KILLS	H	H	H	M	M	M	M	L	M
2.4.7	CORRELATE	H	H	H	H	L	L	L	L	M
2.4.8	INITIATE TRACK	H	H	H	H	L	L	L	L	M
2.4.9	ESTIMATE STATE	H	H	H	H	L	L	M	L	M
2.4.10	PREDICT INTERCEPT & IMPACT POINTS	H	H	H	M	L	L	L	L	M
2.4.11	INTERPLATFORM DATA COMMUNICATION	H	H	H	H	L	L	L	M	H
2.4.12	GROUND-SPACE COMMUNICATION	H	H	H	M	L	M	M	L	M
2.4.13	GROUND COMMUNICATION	M	M	M	M	M	H	M	H	L
2.4.14	ASSESS THREAT	H	H	H	L	L	L	L	L	L
2.4.15	ASSESS SDS	H	H	H	L	L	L	L	L	L
2.4.16	ASSIGN & CONTROL SBI WEAPONS	H	H	H	H	L	L	L	L	M
2.4.17	ASSIGN & CONTROL GBI WEAPONS	H	H	H	M	L	H	L	L	M
2.4.18	GUIDE & CONTROL SBI WEAPONS	H	H	H	H	L	L	L	L	H
2.4.19	GUIDE & CONTROL GBI WEAPONS	H	H	H	M	L	H	L	L	M
2.4.20	COMMAND ENVIRONMENT CONTROL	H	M	H	M	M	H	M	L	M
2.4.21	CONTROL ONBOARD ENVIRONMENT	H	H	H	H	L	L	L	L	M
2.4.22	COMMAND ATTITUDE & POSITION CONTROL	H	M	H	M	M	H	M	L	M
2.4.23	CONTROL ONBOARD ATTITUDE & POSITION	H	H	H	H	L	L	L	L	M
2.4.24	SENSE ONBOARD STATUS	H	H	H	M	L	L	L	L	M
2.4.25	ASSESS STATUS	H	M	H	M	M	H	M	L	M
2.4.26	COMMAND RECONFIGURATION	H	M	H	M	M	H	M	L	M
2.4.27	RECONFIGURE	H	H	H	H	L	L	L	L	M
2.4.28	DEVELOPMENT TOOLS	M	M	M	L	M	M	M	M	M
2.4.29	HWIL SIMULATOR	M	H	H	M	M	M	L	L	M
2.4.30	DEMONSTRATION SIMULATIONS	M	H	M	M	M	M	M	L	M
2.4.31	SUPPORT DEVELOPMENT TEST	M	M	M	M	H	H	H	H	M
2.4.32	PROVIDE DEVELOPMENT ENVIRONMENT	H	M	H	M	H	M	L	L	M
2.4.33	SUPPORT FACTORY TEST	M	M	M	L	H	H	H	H	L
2.4.34	SUPPORT ACCEPTANCE TEST	M	M	M	M	L	M	L	L	M
2.4.35	MAINTAIN & CONTROL MGMT INFO DATABASE	M	M	H	M	H	H	H	H	M
2.4.36	MANAGEMENT INFORMATION TRACKING	M	L	H	L	H	H	H	H	M

0389/002-001

Figure 2.1-2 SDS Software Application Domains and Quality Factors

Open Issues – One major open issue is the establishment of procedures for audit and review activities for the rapid prototype development process. After that is complete, then software metrics application can be integrated with it. A formal methodology for rating quality must be developed. Without that, unstable metrics models can result. Also, a formal methodology to "tune" metrics models must be defined. This includes developing formal quality rating criteria, as well as guidelines for modifying scores. Without this definition, attempts to "trade off" one quality factor against another are not likely to provide the desired results. Finally, software reuse must be aggressively encouraged. Software metrics can provide estimates of software reusability, which can be compared with actual reuse, providing a measure of one of the most promising sources of cost/schedule reduction.

2.1.2 Summary of Recommended Models and Metrics for SDS Software Measurement (TR-9033-2)

The collection of metrics data identified in Subtask 2, TR-9033-2, proved to be both skewed and elusive. Skewed in the sense that much of the existing metrics information that is found to exist and is formalized via models and mathematical relationships is encountered late in the life cycle (i.e., occurring in the implementation phase and beyond). Elusive due to the fact that early life cycle (predictive) metrics are virtually non-existent, and when identified, have no formal foundations (mathematical or relational) to support them for the most part. Furthermore, consensus is still evolving on the valid scopes of specific metrics applications (e.g., complexity) .

The metrics which have been standardized by the Air Force and Army for use throughout the life cycle are a set of primitive indicators and weighted summations for quality and management control and accounting rather than predictions. Also, productivity models have been in wide use, with quite a rich experience base as to fine-tuning parameters; the second generation; of the Constructive Cost Model "COCOMO" has active user groups. However, one message which should be clear out of this study: each program management shop should establish and implement its own metrics methods and quality management program and develop its own expertise with its software measurements and data.

Though there has been no early life cycle predictive model identified, the use of the AFSCP-800-43, -14 indicators will provide early and consistent software productivity and quality statistics. When advanced metrics development proceeds newer models can be introduced with an established SDS software measurement database for subsequent validation. Also in the absence of an algorithmic, predictive model, a multi-attribute, composite or vector metric representation presents a practical approach to communicate and scale an assortment of primitive indicators. This use of multi-attribute modelling ("fuzzy" and simple) has been recently added to the statistics and O-R literature and is the subject of the 1990 International Conference on Metrics being held in the United States for the first time in several years.

Software Quality and Management Indicators – The review of the metrics contained in Subtask 2 reveals that no single metric is capable of supporting decisions across an entire software domain as defined in Subtask 1, TR-9033-1. Each metric supports measurements in one or two software attributes; e.g., Reliability, Maintainability, etc. The implication here, is that perhaps several metrics will need to be combined to form a composite model capable of supporting decisions on the 36 SDS subfunctions or the defined software domains. The analysis concludes that no available metric should be dropped from consideration.

The early use of the Air Force Standard Quality and Management Indicators are recommended. With the advent of the Software Management and Reporting Tool "SMART", the collection and management of the indicators will be much more efficient and flexible. Synopses of the Management and Quality Indicators are presented in Table 2.1.2-1 and 2.1.2-2.

Composite Metrics and Early SDLC Phase Models – TR-9033-2 also proposed the creation of a vector which could be used to assess the relative strength or weakness of a metric when it is applied to one of the 36 SDS subfunctions. The analysis concluded that many of the relative software attributes are not address by any of the available metric models. However, it was noted that current research is being conducted at several universities, in industry and in government. The need to address the measurement of software attributes, early in the software development process, is well known. Several approaches were evaluated during the performance of Subtask 2. A more recent approach is the use of an emerging class of metrics designated as multi-attributes or multi-criteria metrics. Work on this type of metric is currently on-going at universities, some as close as George Mason University in Fairfax, Virginia.

Table 2.1.2-1 AFSCP 800-43 Software Management Indicators

1. Computer Resource Utilization (CRU) Indicator :

1.A Metrics Used:
- Planned deliverable resource (Memory, CPU, I/O).
- Minimum (Proposed to be delivered) deliverable resource.
- Actual utilization.

1.B Users:
- Development contractors.
- Software development project officers.
- Program managers.
- Product division.

1.C SDLC:
- Requirements definition phase.

2. Software Development Manpower Indicator :

2.A Metrics Used:
- Planned and actual staff deviation.
- Staff losses.

2.B Users:
- Development contractors.
- Software development project officers.
- Program managers.
- Product division.

2.C SDLC:
- All phases.

3. Requirements Definition & Stability Indicator :

3.A Metrics Used:
- Total number of software requirements.
- Number of untraceable s/w requirements.
- Number of untestable s/w requirements.

TASC THE SPARTA ARI GRC DSA
THE INTEGRATED SETA TEAM COMMITTED TO SDIO

Table 2.1.2-1 AFSCP 800-43 Software Management Indicators (Continued)

3.B <u>Users:</u>

- Development contractors.
- Software development project officers.
- Program managers.
- Product division.

3.C <u>SDLC:</u>

- Phases : requirements definition, design, test and reviews (i.e. IIR, SRR, SDR).

4. <u>Software Progress-Development and Test Indicator:</u>

4.A <u>Metrics Used:</u>

For the development progress portion

- CSCI design completion

=(units 100% designed/total units pers CSCI)X100

- CSCI unit test completion

=(units 100% coded & tested/total units per CSCI)X100

- CSCI integration completion

=(units 100% integrated into a CSCI/total units per CSC)X100

For the testing progress portion

- s/w test successfully completed.
- Problem reports opened.
- Problem reports closed.

4.B <u>Users:</u>

- Development contractors.
- Software development project officers.
- Program managers.
- Product division.

4.C <u>SDLC:</u>

- Design, code, and test phases.

Table 2.1.2-1 AFSCP 800-43 Software Management Indicators (Continued)

5. Cost/Schedule Deviations Indicator:

   5.A Metrics Used:
   - Cost variance=BCWP-ACWP.
   - Schedule variance=BCWP - BCWS.
   - Cost performance Index(CPI)=BCWP/ACWP.
   - Schedule performance index(SPI)=BCWP/BCWS.
   - Estimate at completion(EAC)
     =ACWP + (BAC-BCWP)/(0.2SPI + 0.8CPI).
   - Variance at completion(VAC)=EAC - BAC.
   - Percent completed=BCWP CUM/BAC.

   5.B Users:
   - Development contractors.
   - Software development project officers.
   - Program managers.
   - Product division.

   5.C SDLC:
   - Reviews at all phases.

6. Software Development Tools Indicator :

   6.A Metrics Used:
   - Number of months available from when the tools are required to when the tools are expected to be delivered.

   6.B Users:
   - Development contractors.
   - Software engineers
   - Software development project officers.
   - Program chief engineers
   - Program managers.
   - Product division.

Table 2.1.2-1 AFSCP 800-43 Software Management Indicators (Continued)

6.C <u>SDLC:</u>

- Probably during design, code, and test phases.

Table 2.1.2-2 AFSCP 800-14 Software Quality Indicators

1. Completeness Indicator :

1.A Metrics Used:

- Weighted sum of components defined by SDLC milestone delivery schedule checklists

1.B Users:

- Development contractor's software engineering, test, and quality organization.
- Software development project officers.
- Program managers.
- Product division.

1.C SDLC:

- During dem/val SRR, test phases.

2. Design Structure :

2.A Metrics Used:

- Design Structure= w D

Where, w = weight associated with each component (0-1).

D = Components defined as boolean or scored rating of structuring policy implementation.

2.B Users:

- Development contractor's software engineering and test organizations.
- Software development project officers.
- Program managers.
- Product division.

2.C SDLC:

- FSD (PDR, CDR) and testing phases.

Table 2.1.2-2 AFSCP 800-14 Software Quality Indicators (Continued)

3. Defect Density Indicator :
   - 3.A Metrics Used:
     - (Cumulative defects encountered)/(total number of units per CSCI).
     - (Cumulative defects corrected)/(total number of units per CSCI)
   - 3.B Users:
     - Development contractor's software engineering, test and quality organizations.
     - Software development project officers.
     - Program managers.
     - Product division.
   - 3.C SDLC:
     - FSD (PDR), and testing phases.

4. Fault Density Indicator:
   - 4.A Metrics Used:
     - Cumulative faults (causes of failures) / total number of units per CSCI.
     - Cumulative faults corrected / total number of units per CSCI.
   - 4.B Users:
     - Development contractor's software engineering, test, and quality organizations.
     - Software development project officers.
     - Program managers.
     - Personnel performing government acceptance of the CSCI.
     - Product division.
   - 4.C SDLC:
     - FSD (CDR), acceptance testing phases.

5. Test Coverage Indicator :
   - 5.A Metrics Used:
     - (No. of implemented capabilities tested)/(total required capabilities)X100%
     - (Software structure tested)/(total software structure)X100%

Table 2.1.2-2 AFSCP 800-14 Software Quality Indicators (Continued)

5.B Users:

- Development contractor's software engineering, test, and quality organizations.
- Government acceptance personnel.
- Software development project officers.
- Program managers.
- Product division

5.C SDLC:

- Functional and requirements testing phases.

6. Test Sufficiency Indicator :

6.A Metrics Used:

- Remaining faults(FR)=(PF - FP)X(UI/UT)
- Maximum tolerance(MAXT)=c_0 (FR)
- Minimum tolerance(MINT)=c_0 (FR)

Where: PF=number of faults predicted.
FP=number of faults detected before s/w integration testing.
UI=number of units integrated.
UT=number of units in the CSCI.
FD=number of faults detected to date during test.
c_0, c_0 are maximum and minimum tolerance coefficients

6.B Users:

- Development contractor's software engineering, test, and quality organizations.
- Government product acceptance personnel.
- Software development project officers.
- Program managers.
- Product division

6.C SDLC:

- s/w integration testing phase.

TASC TBE SPARTA ARI GRC DSA THE INTEGRATED SETA TEAM COMMITTED TO SDIO

Table 2.1.2-2 AFSCP 800-14 Software Quality Indicators
(Continued)

7. <u>Documentation Indicator :</u>

7.A <u>Metrics Used:</u>

- Documentation Index(DI)

Normalized Summation of $w1 \times D + w2 \times S$

where: w1 and w2 are the weights associated with the assessments of the documentation and source listings , respectively.

D is the documentation products.

S is the source listing.

7.B <u>Users:</u>

- Development contractor's software engineering, logistics, and quality organizations.
- Government products acceptance personnel.
- Software development staff officers.
- Logistics staff officers.
- Program managers.
- Product division.

7.C <u>SDLC:</u>

- s/w products acceptance phase.

It should be noted that the result obtained in calculating a multi-criteria metric (i.e., a single scalar value) provides the same information as a single complexity metric value - of limited use. However, it is felt that a composite vector or multi-attribute vector would provide greater insight into the nature of the life cycle activities when properly instrumented as a result of the visible vector components or coefficients that are always present or absent in some cases. Such a requirements maturity or stability vector would be of the form **R** = A**i**+B**j**+C**k**,

where the

- **i** component represents a traceable requirements activity axis (e.g., user, system, performance, derived)
- the **j** component represents a document maturity component axis (e.g., needs document, draft document, final design spec, draft final document)
- and the **k** component represents a design review component axis with related life cyc'e phase information (e.g., system design, software design, test, or PDR/CDR/FDR states).

The attenuation or amplification of the vector components are contained in the matrix coefficients for **i**, **j**, **k**; A, B, C respectively of form:

$[a_{11}\ a_{12} \ldots a_{1n}]$

$[a_{21}\ a_{22} \ldots a_{2n}]$

$[a_{m1}\ a_{m2} \ldots a_{mn}]$

Requirements synthesis vectors in the early life cycle development phases prior to developer source selection activities would be of the form:

R = A**i**+B**j**
or
R = A**i**

A vector result of one of the latter two types in the development phase, after contractor selection, would be of immediate suspect and concern to a program manager since the resultant vector should be of the form A**i**+B**j**+C**k**.

It would be obvious that a development component is missing, possibly as a result of major problems or exceedingly high resource consumption rates.

This is one approach that appears to shed greater information on requirements stability, than multi-criteria metrics and is in its early stages of development. Vector metrics can be plotted, with associated derivatives, to assess work progress or instability points along a critical path. Constant cost curves or surfaces can be generated by examining vector magnitudes to assist in the trade-offs of resource consumption and risk. It is felt that a finer granularity can be achieved over existing cost models with the introduction of vector metrics.

Vector metrics implementation is based on the assumption that predictive measures and measurands can be identified instrumented and automated via a requirements tool set. One such tool set can be identified establishing several possible implementations and a potential series of experiments is with the use of the RVTS tool suite and the predictive measurands identified in Table 2.1.2-1.

2.1.3 Summary of Recommended Tools for SDS Software Measurements

The following subparagraphs summarize the findings from Subtask 3 concerning the tools and environments that support the selected measurement processes and metrics.

Software Management and Report Tool (SMART) – The SMART package is targeted for deployment at the U.S. Army Communications - Electronics Command (CECOM) July-August 1989. The first program it will be applied on is the Advanced Field Artillery Tactical Data System (AFATDS).

The software metrics goal is to implement the Software Management and Quality Indicators as per AFSCP-800-43 and AFSCP-800-14 in an efficient, extensible architecture. The software metrics indicators and data management will be based on IBM PC-compatible machines using the popular "dBase3" formats with the "Clipper" data base language system.

The list below shows some of typical project control data used by SMART to collect the software management and quality indicators.

OVERVIEW OF SMART SOFTWARE DATA COLLECTIONS

Deviations
Waivers
Software Trouble Reports
Test Incident Forms
Engineering Change Proposals
Software Improvement Reports
Software Discrepancy Reports
Computer Resource Utilization
Software Development Manpower
Requirements Definition and Stability
Software Progress - Development and Test
Cost/Schedule Deviations
Software Development Tools
Completeness
Design Structure
Defect Density
Fault Density
Test Coverage
Software Maturity
Documentation

Statistical Modelling and Estimation of Reliability Functions for Software (SMERFS) – The Farr and Smith's SMERFS package of reliability models from the Naval Surface Warfare Center (NSWC) at Dahlgren presents a field tested, efficient tool for the SDS to use on an interim basis. As the interim use of the SMERFS models progresses, concurrent validations and research into highly reliable system requirement issues can be used to tune the models' deployment and to plan for an advanced model. The aspect of applying uncertainty theory should also be examined during this interim period, so that the potentiality of applied uncertainty modelling can be assessed.

SMERFS was developed several years ago as an aid in the evaluation of software reliability. In its original design it was targeted for mainframe and mini-computer environments. Since then it has also been adapted to operate on micro-computers, specifically IBM-PC/XT compatibles.

The current version of SMERFS has incorporated eight software reliability models. The models include the following: (1) Musa's Execution Model, (2) Goel-Okumoto Non-Homogeneous Poisson Model, (3) Adapted Goel-Okumoto Non-Homogeneous Poisson Model, (4) Moranda's Geometric Model, (5) Schafer's Generalized Poisson Model, (6) Schneidewind's Model, (7) Littlewood-Verrall Bayesian Model, and (8) the Brooks-Motley Model.

SMERFS contains a driver which is claimed to make it machine independent. The driver is a subset of the American Standards Institute (ANSI) specifications for the FORTRAN 77 compiler. Several user selectable options are available within the driver and allow the system to be configured to produce: better predictions; output plots and catalogued output files. Currently SMERFS is operational on three main computer groups at the Naval Surface Weapons Center (NSWC), Dahlgren, VA. The three computer groups include the CDC CYBER 170/875, the Vaxcluster 11/785, and a large number of IBM-compatible PCs. Dr. William H. Farr, of NSWC, and Mr. Oliver D. Smith, of EG&G Washington Analytical Services Center, Inc. both claim that transferring SMERFS to other computers should be very easily accomplished.

Besides containing operating instructions within its interactive mode, SMERFS two additional pieces of documentation. The two supplemental reports are: (1) SMERFS Library Access Guide (NSWC-TR-84-371, Rev. 1), and (2) SMERFS User's Guide (NSWC-TR-84-373, Rev. 1). These two publications allow a potential user to preview the system. Examples are provided throughout the User's Guide, allowing a potential user to acquire an overview of the SMERFS processing. In addition, the guide also shows actual software reliability analyses performed on the CDC CYBER 170/875.

The SMERFS systems show tremendous potential for use in the SDI environment with a minimum of modification.

ADC Packages AMS, QES, ASQS – These three programs at the Rome Air Development Center are in different life cycle phases as described.

Automated Measurement System (AMS) – The Automated Measurement System (AMS) was developed by the Harris Corporation for the Rome Air Development Center to implement the RADC Software Quality Metrics as specified in RADC-TR-85-37. The RADC-Harris goal was to interface with prevalent coding languages (Fortran, Ada, COBOL) and specification languages (SREM/RSL, SDDL) on a common computer system platform (DEC VAX, VMS, CMS), written in Fortran, with off-the-shelf (OTS) software reused from prior RADC and contractor activities.

The development was generally successful with a few significant shortfalls. Particularly, the following partial language interpretations are reported as disappointments:

Partial Language Support Percentages

Ada	36%
Fortran	60%
SDDL	46%
SREM/RSL	10%

Verbal reports indicate that the contractors performance testing in regard to the Ada interface has not been satisfactorily duplicated. The final report indicates an unsatisfactory porting to an IBM PC/AT of the AMS. Oral reports indicate that the attempt was totally unsuccessful. The use of AMS by the SDIO for Fortran and some Ada may be helpful in a VAX environment.

Quality Evaluation System (QES) – The RADC program titled Quality Evaluation System (QES) is also called "son of AMS" by some RADC staff. This package will be written in Ada to support both Ada and Fortran applications. The target date is late 1989 or early 1990. Progress of this package should be tracked by the SDIO.

Automated Specification Quality System (ASQS) – The RADC has begun work on an Au.omated Specification Quality System (ASQS) for early SDLC phase reporting. Development of this program should be tracked by the SDIO.

2.2 REVIEW OF FINDINGS AND RECOMMENDATIONS

The SDS software measurement requirements documented in TR-9033-1, and summarized in 2.1.1, were developed with an eye toward refinement and expansions. Similarly, while a vast array of information was amassed and evaluated in completing TR-9033-2, the task was necessarily incomplete, given the resources available and the constantly changing set of new products being developed. Therefore, a critical review of these efforts by other knowledgeable professionals is considered not only prudent, but necessary, to achieve a consensus on the SDS Software Measurement approach. Another purpose in having review of SDS software measurement plans is to more closely involve affected participants in the process. Besides having expert experience in diverse areas, many of the reviewers will be involved in initiating the measurement program and in operating within the requirements of that program. Thus, they will have the opportunity not only to learn abort requirements that may be imposed upon them, but also to shape those requirements. This can be a key factor in the early establishment and success of the program.

The review process has the following steps:

a) Identify target review audiences

b) Group attendees for briefings as desired

c) Prepare briefing material

d) Distribute review material

e) Present briefings and receive immediate feedback

f) Analyze, assimilate feedback

g) Modify SDS Software Measurement approach as necessary

h) Prepare and distribute specific standards, policies, and plans for implementation.

The conclusions contained in TR-9033-1 and TR-9033-2 should be distributed to "interested parties" in the field of SDS software development. They would include: a) the SE&I contractor; b) Software Center personnel; c) SPO personnel; f) procurement personnel; g) software tools and environment developers; h) software measurement academicians; and i) others as deemed appropriate by SDIO.

While not all the reviewers would be interested in all aspects of the reports, the analysis and reasoning in the technical reports would be useful in providing a good background for review. For example, a software measurement academician would primarily be interested in the measurement approach and metrics models; however, the identification of the 36 SDS software application domains would be useful information to know, as domain definition is an integral part of our approach.

The reviewers should concentrate on the following areas, with emphasis on areas that directly affect them:

a) The proposed software typing scheme and resultant candidate generic software domains;

b) The relative importance of the software quality factors within each domain;

c) The proposed baseline methodology for setting specific goals or targets for software quality factors;

d) The metrics models and tools designated as appropriate for each domain and factor or attribute; and

e) The proposed application and feedback processes for measurement of the SDS software.

The reviewers could potentially contribute substantially to the successful definition and use of software measurement in SDS, given that they may have a narrower, but deeper, experience base (e.g., the BSTS contractor, in BSTS software functions). Also, with a good understanding of the measurement program, more volunteers for pilot experiments would be likely.

The material to be reviewed (either existing documents or abstracts) should be distributed to the reviewers in the near future. Concurrent with the distribution, SDS Software Measurement briefings should be developed and given to the reviewers. This approach allows tailoring of the material to a specific audience. For example, briefings to academicians might concentrate on the models chosen, while briefings to major component contractors would emphasize the development of the SDS software application domains. Tailored briefings also allow for receiving immediate verbal feedback. Without the briefings, written feedback, possibly weeks or months later, could be the only response.

Alternatively, if the number of briefings required to address each review group is deemed excessive, briefings could be combined to address several, or perhaps all, reviewers at one time. A simple briefing session (2-3 days) would require considerably more organization time, and probably lead time as well, but would have the advantage of each review group hearing first hand the comments and ideas of the other reviewers. In addition, while set up time would be increased, the overall time to distribute material, present the briefings, and receive feedback could be less than that necessary if multiple, targeted briefings were given. As noted above, one of the key review areas is the set of tools and software metrics appropriate for a project. To better acquaint the audience with the process, a step by step presentation of setting the software measurement goals for a project, and determining the measurements to be taken, should also be given. This is a particularly recommended approach.

Following the briefing(s) and review comments assimilation, the comments should be reviewed for possible incorporation into the SDS Software Measurement program. Software domains should be refined in light of the then-current understanding of the systems functions. Quality factor rankings should be adjusted as necessary, and the software measurement toolset specified. The modified standards, policies, and plans should receive wide distribution, and implementation should begin. We recognize that this is not a one-time effort; it will continue throughout the development effort. However, it is necessary to assemble the program with best and most complete information available at that time, and get it started. Otherwise, a fragmented and partial approach, at best, is the likely alternative.

2.3 DEFINITION AND DEVELOPMENT OF MEASUREMENT SYSTEM

Just as it is very important to get the SDS Software Measurement Program off to an early start, so too should the Measurement System that implements the models and software metrics of the program be initiated immediately. Initially, the Measurement System will be less a complete system than a collection of selected tools as identified in Section 2.1. The emphasis is on off-the-shelf availability, with the more mature tools given preference over beta-release products. Other

TASC TBE SPARTA ARI GRC DSA
THE INTEGRATED SETA TEAM COMMITTED TO SDIO

than perhaps some tailoring of the tools to the SDS Software Development environment, no extensive development effort should be initially undertaken. Experience must be gained with the initial toolset before major additional system enhancements are initiated.

Enhancements will be identified through two parallel paths. First, the gaps identified in TR-9033-2 for tools support of quality measurements must be filled. Those cases where a metric is identified, but no tools implements that metric, should be addressed first. Here, the requirements are more completely specified, and near term system improvements are more readily achievable. For those cases where a measurement requirement has been identified, but no approach has been identified or found to be acceptable, additional research will be necessary (see Sections 3.1 and 3.2). Priority for development of software metrics tools should be given to those metrics which are associated with quality factors ranked "High", "Medium", and "Low", in that order.

The other source of system enhancement requirements is that resulting from use of the measurement system. Following its initial definition, users of the system will identify areas of improvement. Some expected areas would be: a) user interface; b) integration of tools; and c) performance. Integration of the toolset and performance would be particularly critical areas for improvement. If the tools comprising the Measurement System are not well integrated, require duplicative data gathering, or wok at cross purposes, resources will be wasted. Of paramount importance, if the tools themselves do not produce the results expected, they must be revised and improved as soon as possible. Otherwise, the Measurement System's value would be questionable.

As noted in TR-9033-2, most metric tools available today deal with the later parts of the development life cycle (i.e., code). Given the emphasis on the value of predictions of SDS software quality and process factors, automated support for evaluating activities earlier in the life cycle is essential. Thus, while the initial focus of the Measurement System will be on the code itself, the early enhancements should also concentrate those tools that aid in early evaluation, controlling, and predicting of software characteristics. These early qualitative enhancements (spanning more of the development process) may well be more valuable than quantitative enhancements (a deeper or different view of a particular measurement).

2.4 SELECTION OF PILOT CONTRACTS

Given the embryonic nature of the SDS Software Measurement Program, it should not be imposed across all on-going and future software development efforts without initial tryouts. It is important to gain experience with the program under controlled circumstances prior to full-scale introductions. Volunteers should be solicited from among those participating in the Software Measurement Program review. To gain initial experience, a few (two or three) software development efforts should be picked for introduction of the Software Measurement Program. These efforts should be: a) relatively small; b) well-defined; and c) of short duration. Efforts with these characteristics should be picked to allow for rapid assessment of results, and minimization of outside influences.

As noted in TR-9033-1, the software measurement process should be tailored to each specific effort. Initially, the software quality specifications should be defined. This is a cooperative effort among the developer, users, the System Program Office, and quality assurance personnel. Out of this effort is accomplished: a) software criticality definition; b) technology risk assessment; and c) software quality criteria and targets specification. The software product must be assigned to the appropriate software domain, and the associated quality and productivity requirements defined, and refined, as appropriate.

Following the definition and ranking of the quality factors, the appropriate software metrics models and tools would be selected from the available set for application throughout the development process. The contractor then would initiate development, integrating the selected metrics tools and procedures with the development process. The reports and analyzes output from the measurement system would then be reported on at the appropriate points in the review process, allowing for continuing assessment of the software. As results become available, they could be compared with the objectives set during development initiation. Deviations from objectives would be noted, explained, and appropriate action taken to adjust either the objectives, the development process, or both. This process of continuing monitoring, assessment, and adjustment would continue through the life cycle until product delivery. At that time, direct user assessment of the developed system would be compared to the predictors which were developed from application of the software development process. From this comparison, the value of the software measurement process in providing accurate predictors of software attributes would be determined.

It is from this initial application of the Software Measurement Program in the "real-world" that will give SDIO the feedback necessary to determine the future course of the measurement effort. If the results reasonably meet expectations, then additional contracts should be selected for use, and the effort expanded. If, however, the results are not particularly good, then either modifications to the Software Measurement Program must be made, and initial trials re-executed, or the utility of the entire effort may be questioned.

In order to provide a high degree of control and consistency of results in the initial trials, it would be advisable to have the Software Center administer the Measurement Program. As the probable repository of the complete set of tools and environments, they would have the expertise and resources necessary to support the software measurement program's introduction.

2.5 EXPERIMENTATION AND FEEDBACK

Specific experiments to be undertaken include primarily those which address measurement of the software development process. Prediction of size, cost, and schedule are most readily undertaken, and would likely produce the most unambiguous results. Experiments with a COCOMO-based system and SOFTCOST-Ada are recommended for schedule and cost.

For experiments with software quality, software reliability is the factor is best covered by existing metrics. Experiments with SMERFS are recommended to get reliability predictions.

The above experiments should either be performed by the Software Center or by the contractors selected for initial Software Measurement trials.

3. SDS SOFTWARE MEASUREMENT FUTURE APPROACH

This section discusses activities which should be undertaken in the longer term to provide for continued improvement in the SDS Software Measurement process. Key among these are:

a) continued development of models and metrics

b) data collection and experimentation to validate models and metrics

c) development of comprehensive measurement methodology

d) integration of measurement with the SDS software development process

e) definition and development of automated tools and databases.

3.1 SOFTWARE MODELS AND METRICS DEVELOPMENT

There should be a two pronged evolutionary approach to develop the SDS Software Models and Metrics consisting of the following activity groups of equal value:

a. Applying currently available models.

b. Software Metrics Requirements R&D.

3.1.1 Evolutionary Development

An evolutionary approach is typically used when the knowledge base must mature to achieve both long-range and intermediate objectives. These objectives are:

a. To develop the SDS software metrics management and validation data through state-of-the-art experience with off-the-shelf (OTS) package applications.

b. To feed-forward management evaluations and validations into the future.

c. To develop advanced SDS reliability models including factors for subprototype replacement, Ada packaging architecture and functional criticality and tolerance.

d. To develop appropriate models for SDS Integrity, Portability, Usability and Reusability.

e. To develop composite metrics with meaningful weightings and/or vector presentation.

These objectives are approached by means of short-range and long-range activity groups which must interrelate. The short-range activities must feed-forward experience and evaluations. The long-range activities must provide refinements and validations to the day-to-day activities to promote accuracy and effectiveness.

Figure 3.1-1 shows an overview of the activity groups.

3.1.2 Short-Range Activities

The short range activities are needed to provide control and a basis for validations. As discussed in TR-9033-1 and its related briefing, the application of the Air Force Quality and Management Indicators appear to be a simplistic approach to tracking quality and productivity, but will provide a controlled footing for the program. Soon with the advent of the Software Management and Reporting Tool "SMART", the deployment and use of the indicators can be efficiently managed.

Figure 3.1-2 summarizes the short-range activities.

Software Indicators for Preliminary Data – The early use of the software management and quality indicators will provide a beginning for the SDS software database. The efficient application of the Air Force pamphlets, with contractual/SOW enforcement for the data collections will start to build the productivity and quality control data needed for early management control. More importantly, this early step will provide a basis for the preliminary validations of assessment models.

As experience builds, adjustments to SOWs should be planned for additional and refined data collections.

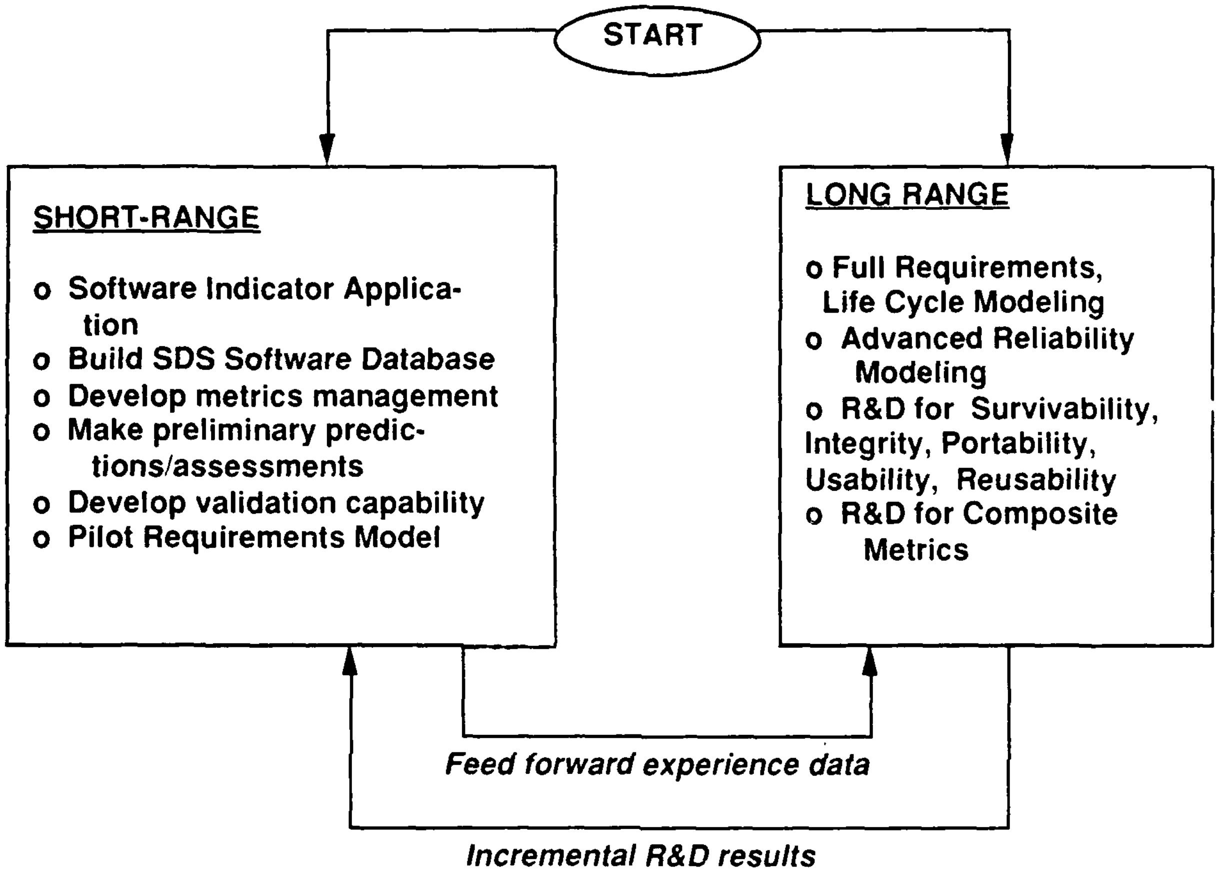

START
SHORT-RANGE
o Software Indicator Application
o Build SDS Software Database
o Develop metrics management
o Make preliminary predictions/assessments
o Develop validation capability
o Pilot Requirements Model
LONG RANGE
o Full Requirements, Life Cycle Modeling
o Advanced Reliability Modeling
o R&D for Survivability, Integrity, Portability, Usability, Reusability
o R&D for Composite Metrics
Feed forward experience data
Incremental R&D results

Figure 3.1-1 Short- and Long-Range Activities

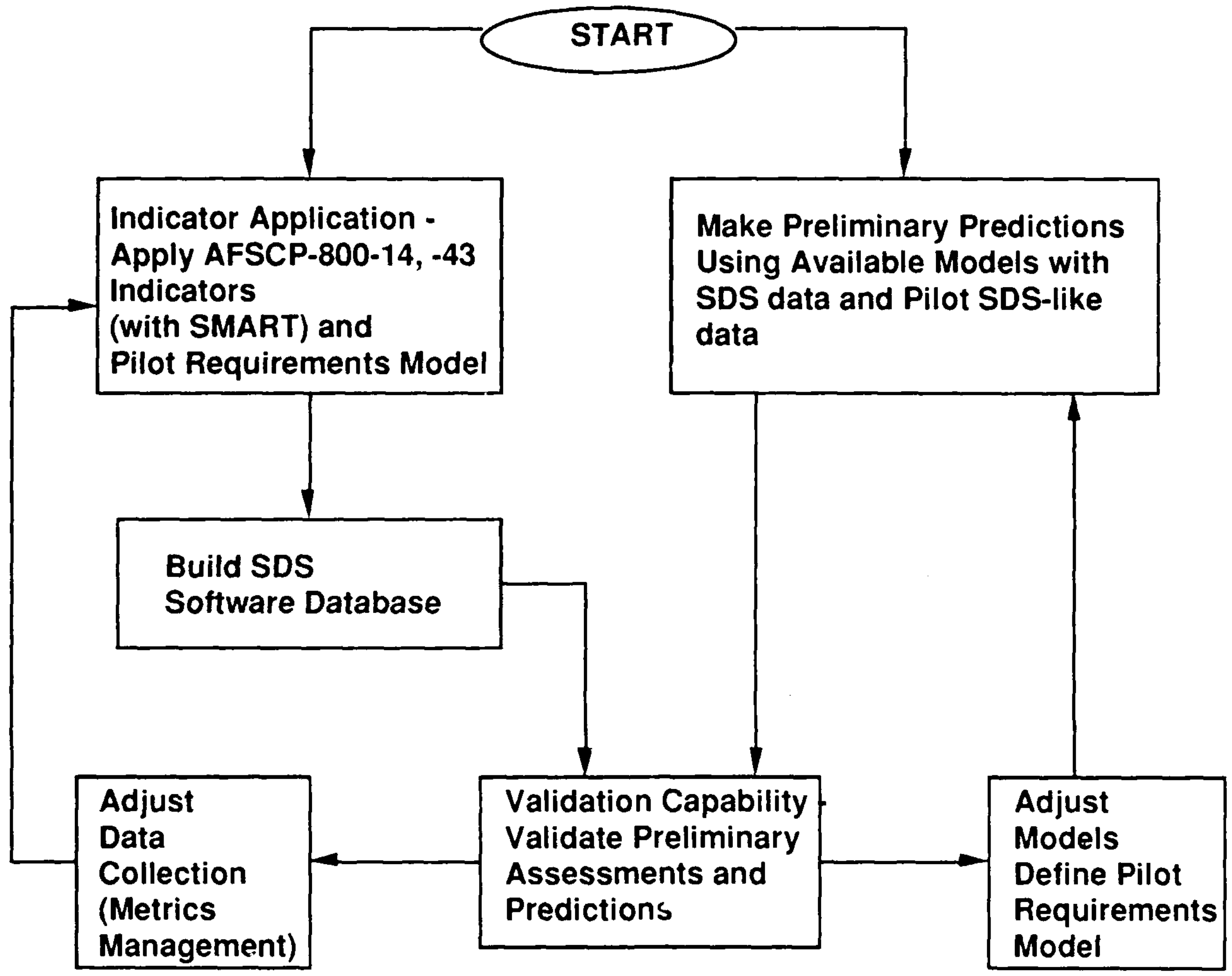

START
Indicator Application - Apply AFSCP-800-14, -43 Indicators (with SMART) and Pilot Requirements Model
Make Preliminary Predictions Using Available Models with SDS data and Pilot SDS-like data
Build SDS Software Database
Adjust Data Collection (Metrics Management)
Validation Capability - Validate Preliminary Assessments and Predictions
Adjust Models Define Pilot Requirements Model

Figure 3.1-2 **Short-Range Activities**

Build SDS Software Database – As soon as practical, the development of an SDS software database should be started. A good starting basis is the data tracked by the indicators package Software Management and Reporting Tool "SMART". A preliminary checklist of software data elements from the early phases of the SDLC is presented in Table 3.1-1.

Table 3.1-1 Preliminary Requirements and Early SDLC Measurands

1.	Requirements Maturity/Requirements Specificity
2.	Level (e.g., System, Subsystem)
3.	Stability/Arbitrariness
4.	Increment Tools Existence
5.	Functional Coupling Rating (Functions Mapped to Requirements)
6.	Derived Requirements Coupling
7.	Requirements Traceability Matrix (yes or no)
8.	Traceability Rating (scored or descriptive)
9.	Requirements Tools Deployment
10.	PTRs/DTRs Opened/Closed by Period
11.	User Qualification/Validation Ratings
12.	Ratio of Validated Requirements to TBDs, Unknowns and Preliminaries
13.	Prototype or Production
14.	IV&V Verification Ratings of Functional Definitions
15.	Development, QA & IV&V Staffing Actual/Budget Ratios
16.	System Spec. Maturity Score
17.	ROC/SOW Mapping Scores
18.	Percentage Replacement/New & Reuse/Unique
19.	Development Environment Completion Rating
20.	Conflicting Requirements Weighted Scoring
21.	Support Requirements Maturity & Completion
22.	Requirements Change Drivers

Preliminary Predictions – In parallel with the deployment of the software quality and management indicators and database development, the program should start the preliminary use of available models. The early use of the Statistical Modeling of Reliability Functions for Software (SMERFS) eight models with SDS data and with borrowed data from similar projects will provide the type of practical experience needed for later refinements and validations. Also, the use of the SECOMO and REVIC project schedule and cost modeling packages will provide management with pilot experience needed to develop an efficient management and control organization.

Both the cost/schedule data with COCOMO/SECOMO/REVIC and the reliability data with the reliability models should be organized and preserved for subsequent validations and model refinements.

Preliminary Validations and Adjustments – Based upon the developing database of SDS software data, the preliminary predictions and assessments can be validated. This activity of preliminary validations may have to continue using borrowed data until enough SDS data are collected but the "learn as you go" aspect of software metrics application must be duly considered.

Pilot Requirements Model – As a necessary aid to provide a timely foundation for structuring early measurands from the SDS life cycle, a requirements model has been proposed. Figure 3.1-3 presents this model.

This proposed requirements model is identified that can also serve as the front-end to whatever life cycle model is subsequently developed consistent with Mil-Std-2167A. The requirements model can serve to establish formal relationships between measurands and be used to establish formal metrics in assessing development risk, and predicting cost/schedule effectiveness.

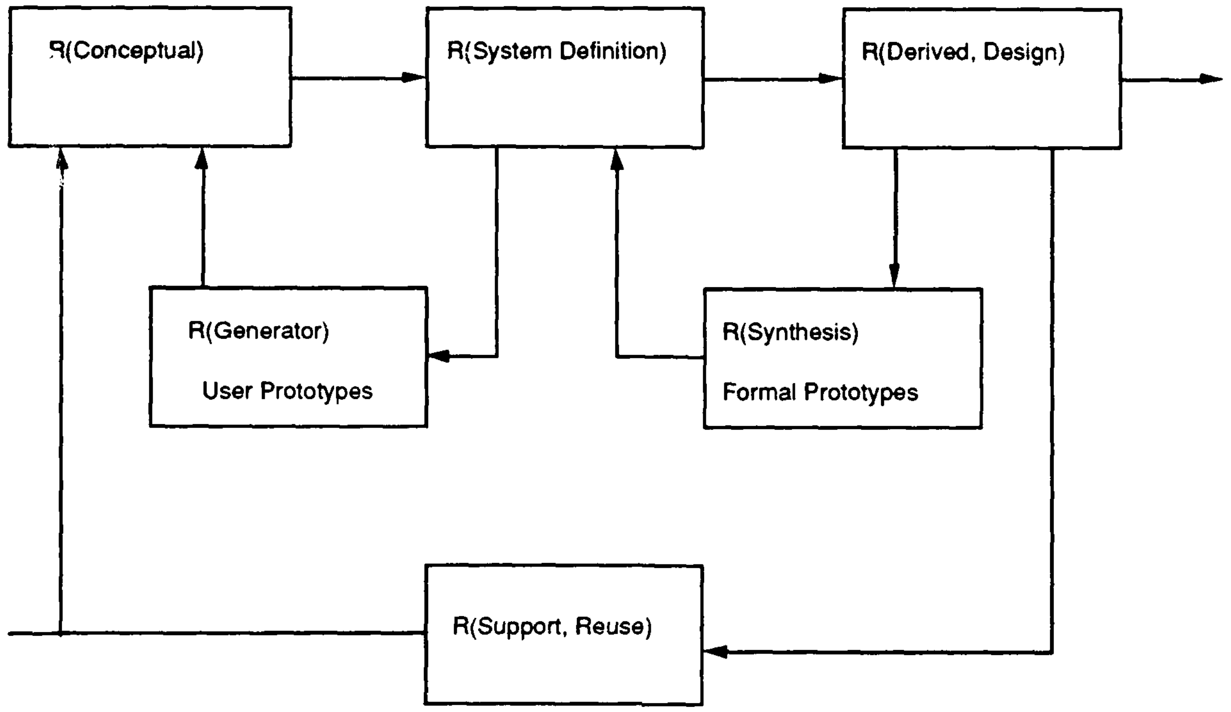


Figure 3.1-3 Pilot Requirements Model

3.1.3 Long-Range Development

As the experience and database with the available models and metrics matures, the importance of advanced models with actual SDS factors and criteria grows. The activities planned for long-range development are:

a. Full SDS Requirements Modeling

b. Advanced Reliability Modeling

c. Survivability, Integrity, Portability, Usability and Reusability Modeling

d. Composite Metrics Modeling

These activities are discussed in the following paragraphs.

Full SDS Requirements Modeling – This activity will start with the pilot requirements model and use a formal method to represent the SDS. This model should be readily extensible into the software architecture and detail design. The model should include the attributes presented in Table 3.1-2.

The pilot requirements model proposed (Figure 3.1-3) has a robust set of requirements generators and the ability to serve as the front-end to diverse types of life cycle models (iterative and non-iterative). The model allows requirements parsing to enable the proper allocation and instrumentation of requirements for subsequent metrics measurements and assessments. Several dozen newly identified measures and measurands (Table 3.1-1) have been identified. These are categorized as predictive measurands and are intended to provide early visibility into requirements synthesis, maturity and stability. The latter two are identified as predictive indicators that can serve to establish acceptable threshclds capable of supporting design. The identification of too many unstable and immature requirements preclude design activities, increases development risk and creates an incomplete design envelope).

To support the model, it is also proposed that formal mathematical relationships between model components be identified. An initial attempt at formal mathematical relationship has been accomplished in conjunction with George Mason University.

Table 3.1-2 SDS Requirements Model Attributes

Multi-level Requirements Identification -- Computer-assisted requirements identification in a separable document basis.

Iterative Requirements Accounting -- Computer-generated matching of low-to-high with high-to-low trace completion reporting.

Incremental Baseline Reporting -- Computer-assisted baseline identification with computer-generated baseline accounting.

Full Real-Time, Data Driven Requirements -- Computer-assisted definition of all required timing constants, data paths and anticipated control paths with computer generated diagnostics and data dictionary.

Hierarchical Decomposition of System Elements -- Computer-assisted hierarchy definition; computer generated hierarchy reporting.

User-Efficient, Secure Storage and Retrieval -- Easy to use menus, helpful messages, meaningful diagnostics with fully supported configuration management.

Functional Simulation/Modeling -- Through computer workstation generated database of the system functions - computer-assisted system modeling and simulation studies.

Document Generation -- Computer workstation generation of system functional documents.

Ada Language Orientation -- Computer generated prototype code or computer-assisted integration of design and code elements with requirements and design specifications.

Life Cycle Meta-Model – A life cycle meta-model (Figure 3.1-4) is required to support the coupling of the requirements model with prototyping and reuse life cycles, as well as technology or automation-based paradigms. The life cycle meta-model is proposed to illustrate a feasible approach to support the various types of development approaches (e.g., evolutionary, incremental, software first). The meta-model is intended to provide the framework for the measurement of productivity, the identification of associated products relative to the measurement of processes, and the elaboration of new design review form factors.

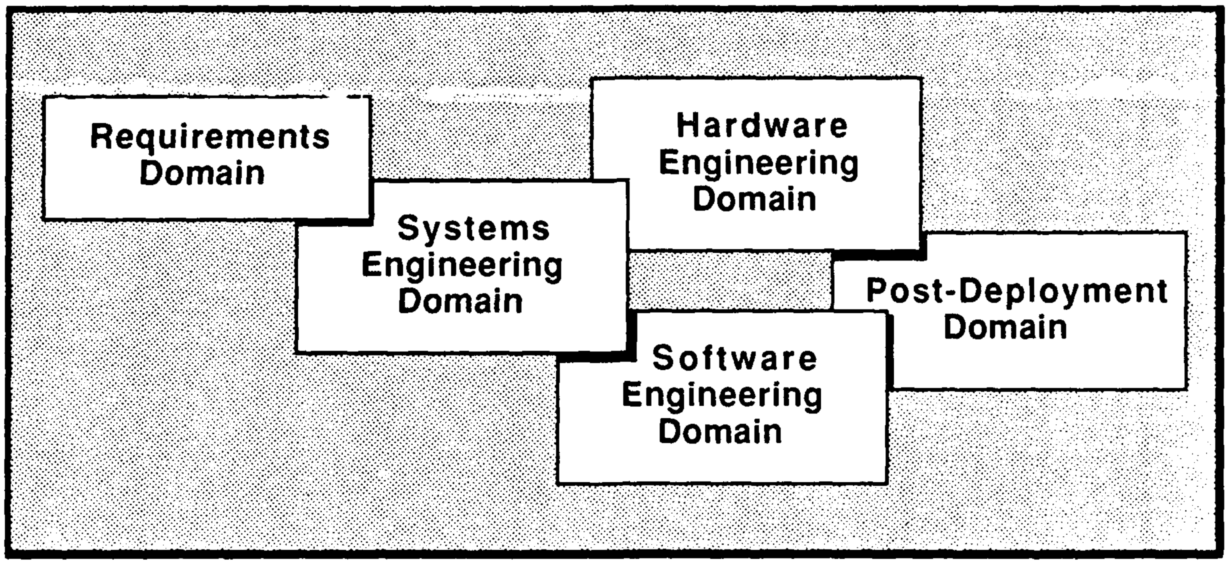

Requirements Domain
Hardware Engineering Domain
Systems Engineering Domain
Post-Deployment Domain
Software Engineering Domain

Figure 3.1-4 Life Cycle Meta-Model

The life cycle building blocks of Figure 3.1-4 are used to map phases of one life cycle form into another, similar to that accomplished in TR-9033-2 for different waterfall models and variations. The model is also required to assist in the association and parsing of categorization data of Table 1.6-6 of report TR-9033-2 across the resulting new life cycle phases.

Advanced Reliability Modeling – TR-9033-2, Paragraph 4.2 described potential shortfalls in available reliability models for the SDS. In particular, the lack of criticality factors and the treatment of all defects as unpredictable were discussed. On a positive note, Paragraph 1.1.6 of that report relayed recent extensions to domain-based model to include the notion of error tolerances in user service. Further development to include scored tolerance would also include the concept of criticality (lack of tolerance).

In a recent article, Jeanette Wing and Mark Nixon have forecast that further extensions to the "Ina Jo" formal specification language may include properties such as "fault tolerance, reliability, performance and real-time behavior." These extensions may be applicable for the SDS reliability, survivability and integrity modeling.

Another valuable extension for SDS is to add software metrics data instrumentation to a formal design language (e.g., "IORL" from Teledyne Brown Engineering), and then use SDI project data (such as SIE and N-SITE) to make experimental coupling and complexity assessments, and selected predictions. The experimental data can be used to generate preliminary norms for SDS predictions and assessments.

Survivability, Integrity, Portability, Usability, Reusability Modeling – These factors require fresh development. Each of these areas was determined to be poorly supported by metrics without formal models, (as represented by TR-9033-2 Appendix B charts 01-36).

Research into these models should be promoted as feasible with priority upon Survivability, Integrity and Reusability.

Composite Metrics – The development of multivariate vector metrics has been applied to the field of general systems optimization modeling.

3.2 VALIDATION OF ADDITIONAL MODELS AND METRICS

The development of new and refined models sets the basis for requirements for a validation testbed or suite consisting of: a) validation methodology base, b) SDS software data and selected test data, and c) a validation tool set. The validation base should provide refinements guidance to further development.

3.2.1 Validation Methodology

The first part of the validation suite is the methodology base. These are at least two exemplary studies providing guidance for software reliability validations Abdel-Ghaly, Chan and Littlewood compared the predictive quality of ten reliability models using multiple criteria (u-plot, y-plot, scatter plots, noise measures, and Prequential likelihood). John Bowen ran goodness-of-fit "GOF" convergence and reasonableness tests upon the eight reliability models in the SMERFS package. These statistical examples should serve as starting methods for the validation methodology.

Prior studies of complexity models also must be considered with the SDS metrics validation methodology. Virginia Gibson and James Senn used simple comparisons to rank six different complexity models. John Stephen Davis and Richard J. LeBlanc conducted tests of nine different complexity measures using three different programming environment databases. Dennis Kafura and Geereddy R. Reddy used simple comparisons to study seven different complexity models.

By establishing an eclectic, extensible methodology, the cost of methodology development can be kept low and yet the responsiveness to the validation task can be ensured.

3.2.2 SDS Software Data and Selected Test Data

Initially the data to use for validations may be obtained from SDI rapid deployment experiments (e.g., SIE and N-SITE). Additional project data may be borrowed from the Rome Air Development Center (RADC) the Joint Surveillance System (JSS), the Naval Underwater Systems Command (NUSC) or other defense systems agencies with similar programs.

Later, as the SDS preliminary application of metrics proceeds, the database will contain prior SDS software metrics life cycle phase data (e.g., error rates, productivity levels).

3.2.3 Validation Tool Set

Two widely used statistics packages are SPSS and SAS. Either offers a PC-based version with interfaces to mainframe services. Alternatively, a basic spreadsheet package like "Lotus 1-2-3" or "Excel" provides a simple tool set to make iterative comparisons of test data and statistical derivations.

Either the statistical packages or the spreadsheet packages have graphics capabilities. The former packages produce selected statistical functions much more effectively, but the spreadsheet packages can make most types of popular graphs (pie, bar, line, etc.).

3.3 COMPREHENSIVE MEASUREMENT METHODOLOGY DEVELOPMENT AND INTEGRATION

Section 3.0 of TR-9033-1 documents the methodology requirements for software metrics. The initial methodology requirements presented there emphasize the need for early introduction of software measurement in the software development life cycle. Just as important, the software measurement process should not be "tacked on" to the development methodology, but integrated with it. Inherent in the integration of the software measurement methodology with the software development methodology are well-developed review points. With the information presented at these review points, not only can redirection of design be undertaken, but also redirection to meet quality goals.

With the above points in mind the following tasks need to be completed:

a) Definition of a standard prototyping methodology for SDS software development.

b) Integration of the standard prototyping methodology with the Software Measurement Program.

c) Development of a standard methodology for rating quality.

d) Development of a standard software metrics model "tuning" methodology.

e) Development of a standard metrics selection methodology.

It is envisioned that prototyping activities (including Rapid Prototyping) will be a vitally important component of SDS development. Today, the prototyping approach is not well supported by standards and guidelines, as is the classic water fall development process. Standards and guidelines are necessary to specify the objectives of each development phase, as well as the corresponding audit and review activities. As noted in TR-9033-1, prototyping comes in at least two types: Throw-away, and Evolutionary. Both types, and intermediate variations, must be accommodated. Specifically to be addressed is the consideration that contractors may already have differing prototype approaches in place, and may be reluctant to change them.

As part of defining the prototyping methodology for SDS, software metrics should be included. The integration would provide a methodology analogous to that for the waterfall model, with software metrics auditing and review activities occurring concurrently with standard audit and review activities. One of the intriguing software metrics proposed is a metric that predicts when prototyping will be complete, and further iterations or modifications to the prototype are no longer necessary or cost-effective.

Current approaches to rating the quality attributes of a software product, including those for the rating criteria, are poorly defined. To achieve uniform ratings among different users, and to obtain ratings which are reliable measures of the extent to which a software product meets its quality requirements, a well-defined rating methodology is necessary. It is also required for tuning the quality metrics, so that their predictive capabilities can be made as accurate as possible.

The validation of software metrics require that they produce outputs, or scores, which can predict the quality rating of the final product with acceptable accuracy and consistency. Validation of a software metric is complex, and requires tuning the model to improve its predictive accuracy until it is acceptable. This tuning process must be repeated across each individual software domain. Because tuning could require modification of literally hundreds of different elements, metrics, criteria, rankings, etc., possibly in combination, it cannot be accomplished by any straightforward method. It is therefore necessary to establish a methodology for software metrics tuning so that convergence to a stable configuration of the metrics can be achieved as expeditiously as possible. The methodology may be both analytical and experimental. Analytical methods may be used for identifying strategies, based on the analysis of the correlations among the quality factor scores, the quality ratings, the applicable metrics, and metric elements. Experimental methods would be used to analyze the effects of the tuning strategies on a variety of software programs within the same software domain. Selection of the tuning strategies may be based on:

a) analytical methods;

b) random searches in the metrics space;

c) past experience.

When a development effort is begun, the selection of the appropriate quality goals, factors, rankings, and tools could be a daunting prospect. Consequently, the guidelines and recommendations for approaches documented in TR-9033-1 need to be formalized into a system that will aid in the selection of the proper software measurement parameters. While the system should eventually be automated, even a manual implementation of a selection methodology would be a significant step forward. The methodology would guide the user in first setting the appropriate quality goals for the development. It would assist in determining the software domain(s) in which the development best fits. Through analysis of criticality level, and tradeoff of related software metrics, the system would help the developer in defining a baseline set of software metrics for his use. While such a system would always require final determination of the selections based on overall considerations, such a formalized approach would remove much of the guesswork from the software metrics selection.

3.4 DEFINITION AND DEVELOPMENT OF AUTOMATED TOOLS AND DATABASES

The process of tailoring the selected tool packages (SMERFS, SMART, SECOMO and/or REVIC) into a smoothly operated, user-friendly management environment is described in this section.

Figure 3.4-1 presents an overview of the activities to develop (or later, enhance) the tools environment.

3.4.1 Tool Acquisitions and Installations

Each of the tools must be properly acquired and installed. Specific setup procedures must be checked. Options must analyzed. Alternative installation sequences should be tested. This step can be ad hoc based upon tool and resource availabilities.

A tentative order to bring in tools is:

a. SMERFS

b. SECOMO or REVIC

c. SMART

d. AdaMat or AdaCAT

e. Others

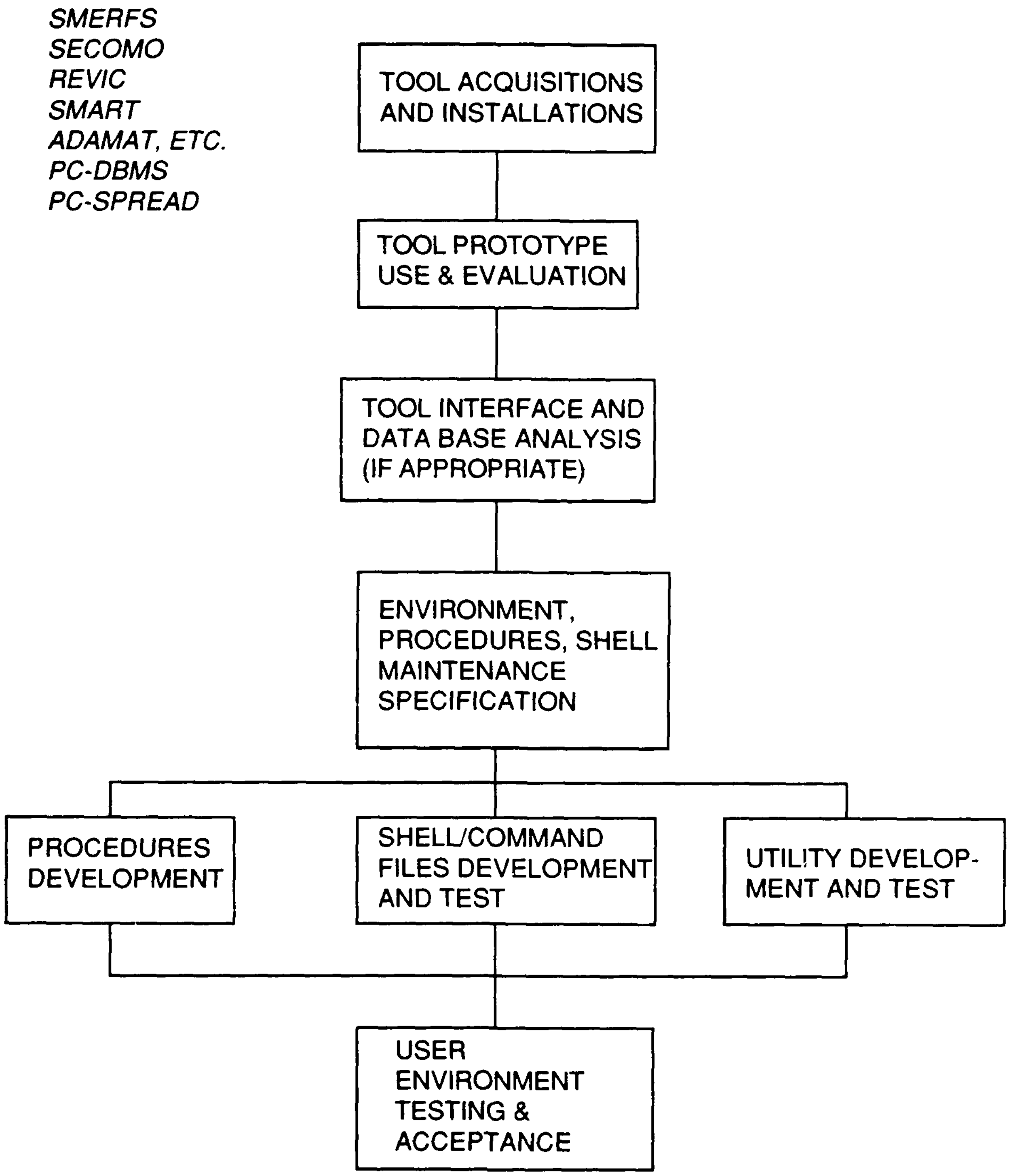

SMERFS
SECOMO
REVIC
SMART
ADAMAT, ETC.
PC-DBMS
PC-SPREAD
TOOL ACQUISITIONS AND INSTALLATIONS
TOOL PROTOTYPE USE & EVALUATION
TOOL INTERFACE AND DATA BASE ANALYSIS (IF APPROPRIATE)
ENVIRONMENT, PROCEDURES, SHELL MAINTENANCE SPECIFICATION
PROCEDURES DEVELOPMENT
SHELL/COMMAND FILES DEVELOPMENT AND TEST
UTILITY DEVELOP-MENT AND TEST
USER ENVIRONMENT TESTING & ACCEPTANCE

Figure 3.4-1 Software Measurement/Environment Tool Development Activities

3.4.2 Tool Prototype Use and Evaluation

As each tool is successfully installed, it can then be scheduled for prototype use, data acquisition and evaluation with sample SDS data. For some tools, e.g., SMERFS, it may be necessary to borrow or fabricate test data. For others e.g., SECOMO/REVIC live data may be applicable.

A preliminary use and evaluation method will be drafted for SMERFS, SECOMO/REVIC and SMART. Later this method will be reviewed for expansion or revisions. Some preliminary evaluation criteria include:

a. Tool input requirements availability

b. User friendliness

c. Report readability

d. Processing cycle responsiveness

e. Adequacy of data integrity controls

Based upon the specific evaluations for each tool, selected utility command files and/or utility programs may be indicated. For example, the SMERFS package provides subroutine interface descriptions to allow a tailored user-specific executive. Alternatively, the off-the-shelf. Executive may only require its incorporation within a menu-based software selection package like MicroSoft Windows or a customized menu written in Clipper, dBase Pascal or C.

3.4.3 Tool Interface and Database Analysis

For those tools requiring selected enhancements, one or more of the following may be required:

a) Command & Parameter Analysis -- a very simplistic analysis for program package which can be driven from a menu handler.

b) Program Interface Analysis -- a more intense analysis to build or adapt an executive module or a low-level hardware driver to new user or hardware demands.

c) Database/File Format Analysis -- a moderate analysis to develop or enhance data extraction services downstream of a package (e.g., SMART to Lotus or to Excel).

3.4.4 Environment, Procedures and Shell Maintenance or Specification

This activity initially may be a modest programming activity to install a menus package. Later, when it becomes necessary to add tools or to adjust the tool selections, the changes should be very brief to make and document.

3.4.5 Procedures Development

As each tool set is integrated into the software metrics environment, the set of user procedures should be drafted. As each tool becomes used, the procedures should be reviewed and revised if needed.

3.4.6 Shell Command Files Development and Test

The computer commands to provide user efficiency and smooth integration must be built and tested by technical support staffs. The data used to test each tool should be preserved for user tutorial sessions.

3.4.7 Utility Development and Test

As appropriate, utility programs should be coded in a popular language like Turbo Pascal or Turbo C. Simple testing should precede complex tests.

3.4.8 User Environment Testing and Acceptance

After each of the models has been unit-tested with its applicable adaption data, the total environment must be addressed. The users will be asked to pass judgement on the smoothness of program initiation and execution.

REFERENCES

REF.1

ALPHABETICAL DOCUMENT INDEX

[ABDE8609]	Abdel-Ghaly, Chan & Littlewood; Evaluation of Competing Software Reliability Predictions; IEEE Transactions, 9/86.
[ADKI7704]	Adkins & Pooch; Computer Simulation: A Tutorial; Computer, 4/77.
[ALBR8411]	Albrecht, A.J.; AD/M Productivity Measurement and Estimate Validation; IBM CIS&A Guideline 313; 11/1/84.
[ALBR8311]	Albrecht & Gaffney; Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation; IEEE Transactions, 11/83.
[ARMY8408]	U.S. Army/Computer Systems Command; Software Quality Engineering Handbook; Subcontract No. 7202, 8/84.
[ARNO86]	Arnold, Robert S.; Tutorial: Software Restructuring; IEEE Computer Society Press, 1986. Includes [YAU8011], [HARR8209]
[BALZ8311]	Balzer, Cheatham & Green; Software Technology in the 1990's: Using a New Paradigm; Computer, 11/83.
[BASI8311A]	Basili & Hutchens; An Empirical Study of a Syntactic Complexity Family; IEEE Transactions, 11/83.
[BASI8709]	Basili & Rombach; Implementing Quantitative SQA: A Practical Model; IEEE Transactions, 9/87.
[BASI8703]	Basili & Rombach; TAME: Tailoring an Ada Measurement Environment; Proceedings from 5th National Joint Ada Conference, 3/87.
[BASI8311B]	Basili, Selby, Phillips; Metric Analysis & Data Validation Across Fortran Projects; IEEE Transactions; 11/83.
[BEHR8311]	Behrens, Charles; Measuring the Productivity of Computer Systems Development Activities with Function Points; IEEE Transactions, 11/83.
[BEIZ84]	Beizer, Boris; Software System Testing and Quality Assurance; Van Nostrand Reinhold Co., 1984.
[BELZ8603]	Belz, Frank C.; Applying the Spiral Model: Observations on Developing System Software in Ada; Proceedings from 4th National Ada Conference, 3/86.
[BEND8703]	Bendifallah & Scacchi; Understanding Software Maintenance Work; IEEE Transactions, 3/87.

TASC THE SPARTA ARI GRC DSA
THE INTEGRATED SETA TEAM COMMITTED TO SDIO

[BOEH81]	Boehm, Barry W.; Software Engineering Economics; Prentice-Hall, New Jersey, 1981.
[BOEH8810]	Boehm & Papaccio; Understanding and Controlling Software Costs; IEEE Transactions, 10/88.
[BOWE8610]	Bowen, John B.; Application of a Multi-Model Approach to Estimating Residual Software Faults and Time Between Failures; Quality & Reliability Engineering International, Vol. 3; 10/22/86.
[BRIT8811]	Britcher, R.N.; Using Inspections to Investigate Program Correctness; Computer, 11/88.
[BROW8407]	Browne, J.C.; Understanding Execution Behavior of Software Systems; Computer, 7/84.
[CARD8508]	Card, Page & McGarry; Criteria for Software Modularization; Proceedings IEEE Conference on Software Engineering; 8/85. (see SEI section)
[CARD8602]	Card, Church & Agresti; An Empirical Study of Software Design Practices; IEEE Transactions; 2/86. (see SEI section)
[CARD8707]	Card, McGarry & Page; Evaluating Software Engineering Technologies; IEEE Transactions, 7/87.
[CARD8709]	Card, Cotnoir, Goorevich; Managing Software Maintenance Cost & Quality; Proceedings IEEE Conference on Software Maintenance, 9/87. (see SEI section)
[CARD8812]	Card & Agresti; Measuring Software Design Complexity; Journal of Systems and Software, 12/88. (see SEI section)
[CARD87]	Card & Agresti; Resolving the Software Science Anomaly; Journal of Systems and Software, 1987. (see SEI section)
[CARD8807]	Card, David; The Role of Measurement in Software Engineering; Proceedings 2nd IEE/BCS Software Engineering Conf., 7/88. (see SEI section)
[CARR8603]	Carrio, Miguel A.; The Technology Life Cycle and Ada; Proceedings from 4th National Ada Conference, 3/86.
[CHAL87]	Chalam, V.V.; Adaptive Control Systems - Techniques & Applications; Marcel Dekker, Inc., 1987.
[CHEU8706]	Cheung & Li; An Empirical Study of Software Metrics; IEEE Transactions, 6/87.
[CHIE8607]	Chien, Y-T; Expert Systems in the SDI Environment; Computer, 7/86.

[CHUB89R]	Chubb, Bruce; Lessons Learned in the Setup and Management of Productivity Improvement Methods.
[CINL75]	Cinlar, Erhan; Introduction to Stochastic Processes; Prentice-Hall, Inc. 1975.
[COHE85]	Cohen, Paul R.; Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach; Pitman Publishing, Inc., 1985.
[CONT86]	Conte, Dunsmore, Shen; Software Engineering Metrics & Models; Benjamin/Cummings Publishing Co.; 1986.
[DAVI8810]	Davis, Bersoff, Comer; A Strategy for Comparing Alternative Software Development Life Cycle Models; IEEE Transactions, 10/88.
[DAVI8809]	Davis & LeBlanc; A Study of the Applicability of Complexity Measures; IEEE Transactions, 9/88.
[DELO8807]	DeLoach, Scott A.; An Interface-Based Ada Programming Support Environment; ACM Press - Ada Letters; July/Aug. 1988.
[DEMU8807]	Demurjian & Hsiao; Towards a Better Understanding of Data Models Through the Multilingual Database System; IEEE Transactions, 7/88.
[DOD8807A]	Department of Defense; Report of the Defense Science Board Task Force on Military Software-9/87; ACM Press - Ada Letters; July/Aug. 1988.
[DOD8807B]	Department of Defense; Ada Board Response to Report of Defense Science Board Task Force on Military Software-2/88; ACM Press - Ada Letters; July/Aug. 1988.
[DORN75]	Dorny, C. Nelson; A Vector Space Approach to Models & Optimization; John Wiley & Sons; 1975.
[DUNS8805]	Dunsmore, H.E.; Evidence Supports some Truisms, Belies Others; Quality Time-IEEE Software, 5/88.
[ELLI88]	Ellis & Wygant; A System-Oriented Methodology to Support Software Testability; ITEA Journal, 1988.
[EVAN84]	Evangelist, Michael; An Analysis of Control Flow Complexity; COMPSAC Proceedings, 1984.
[EVAN83]	Evangelist, Michael; Software Complexity Metric Sensitivity to Program Structuring Rules; Journal of Systems & Software; 1983.
[FARR88]	Farr, Smith & Schimmelpfenneg; A PC Tool for Software Reliability Measurement; Proceedings from Institute of Environment Sciences; 1988.
[FARR8812A]	Farr & Smith; Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) User's Guide; NSWC-TR-84-373, 12/88.

[FARR8812B]	Farr & Smith; Statistical Modeling and Estimation of Relability Functions for Software (SMERFS) Library Access Guide; NSWC-TR-84-371, 12/88.
[FARR8309]	Farr, William H.; A Survey of Software Reliability Modeling and Estimation; NSWC-TR-82-171; 9/83.
[FREE87]	Freeman, Peter; Tutorial: Software Reusability; IEEE Computer Society Press, 1987.
[GADO88]	Gados, Ronald; Guidelines for Certification of Strategic Defense System Simulation Models; ITEA Journal, 1988.
[GELP8806]	Gelperin & Hetzel; The Growth of Software Testing; Communications of the ACM, 6/88.
[GIBS8903]	Gibson & Senn; System Structure and Software Maintenance Performance; Communications of the ACM, 3/89.
[GOIC89RA]	Goicoechea, Burden, Sanchez; Evaluation & Ranking of Alternate Systems with Ariadne: An Application to the Economy Car Selection Problem. (see GMU Info. section)
[GOIC81]	Goicoechea, Hansen & Henrickson; A Hierarchical-Multilevel Approach to the Development & Ranking of Systems in Tanning Industry; Computing & Industrial Engineering, 1981. (see GMU Info. section)
[GOIC89RB]	Goicoechea, Ambrose; Metrics and Trade-Off Analysis in Software Design, Production, and Evaluation.
[GOIC89RC]	Goicoechea, Ambrose; Software Metrics: Their Measurement and Use in Models. (see GMU Info. section)
[HARR89RA]	Harrison, Warren; Applying McCabe's Complexity Measure to Multiple-Exit Programs. (see PSU Info. section)
[HARR8209]	Harrison, Magel, Kluczny, DeKock; Applying Software Complexity Metrics to Program Maintenance; Computer Magazine, 9/82. (see PSU Info. section)
[HARR86]	Harrison & Cook; Are Deeply Nested Conditionals Less Readable?; Journal of Systems and Software, 1986. (see PSU Info. section)
[HARR89RB]	Harrison & Magel; A Complexity Measure Based on Nest'ng Level. (see PSU Info. section)
[HARR88]	Harrison, Warren; MAE: A Syntactic Metric Analysis Environment; Journal of Systems and Software; 1988. (see PSU Info. section)
[HARR87]	Harrison & Cook; A Micro/Macro Measure of Software Complexity; Journal of Systems and Software; 1987. (see PSU Info. section)

[HARR8602] Harrison & Cook; A Note on the Berry-Meekings Style Metric; Communications of the ACM, 2/86. (see PSU Info. section)

[HARR89RC] Harrison, Warren; Software Science and Weyuker's Fifth Property. (see PSU Info. section)

[HARR8804] Harrison, Warren; Using Software Metrics to Allocate Testing Resources; Journal of Management Information Systems, Spring '88. (see PSU Info. section)

[HENR8109] Henry & Kafura; Software Structure Metrics Based on Information Flow; IEEE Transactions, 9/81.

[HUMP8703] Humphrey, Watts S.; Software Process Management; 9th International Conference on Software Engineering, 3/87.

[IDA8804A] IDA-Deliverable to SDIO; Strategic Defense Initiative Architecture Dataflow Modeling Technique, Version 1.5; Report # P-2035, 4/88.

[IDA8804B] IDA-Deliverable to SDIO; A Simple Example of an SADMT Architecture Specification, Version 1.5; Report # P-2036, 4/88.

[IDA8812] IDA-Deliverable to SDIO; SDS Software Testing & Evaluation: A Review of the State-of-the-Art in Software Testing and Evaluation; Report # P-2132, 12/88.

[JEFF8707] Jeffery, D. Ross; Time-Sensitive Cost Models in the Commercial MIS Environment; IEEE Transactions, 7/87.

[KAFU8703] Kafura & Reddy; The Use of Software Complexity Metrics in Software Maintenance; IEEE Transactions, 3/87.

[KARI8812] Karimi & Kuo; User Interface Design from a Real Time Perspective; Communications of the ACM, 12/88.

[KAVI8710] Kavi, Buckles and Bhat; Isomorphisms Between Petri Nets and Dataflow Graphs; IEEE Transactions, 10/87.

[LEAC8703] Leach, Ronald J.; Ada Software Metrics and Their Limitations; Proceedings from 5th National Joint Ada Conference, 3/87.

[LEE8710] Lee, Tony; An Information-Theoretic Analysis of Relational Databases--Part I: Data Dependencies and Information Metric; IEEE Transactions, 10/87.

[LEW8811] Lew, Dillon & Forward; Software Complexity and its Impact on Software Reliability; IEEE Transactions, 11/88.

[LICH8601] Lichtman, Zavdi; Generation and Consistency Checking of Design and Program Structures; IEEE Transactions, 1/86.

[MART8407]	Martin & Brice; Effect of Hardware-Software Interaction on Performance; Computer, 7/84.
[MCCA7612]	McCabe, Thomas; A Complexity Measure; IEEE Transactions, 12/76.
[MCCL7805]	McClure, Carma; A Model for Program Complexity Analysis; Proceedings from 3rd International Conference on S/W Engineering; 5/78.
[MCKA89R]	McKay, Charles; A Proposed Framework for the Tools & Rules to Support the Life Cycle of the Space Station.
[MEND8205]	Mendis, Kenneth S.; Quantifying Software Quality; Computers, 5/82.
[MILL7612]	Mills, Harlan D.; Software Development; IEEE Transactions, 12/76.
[MUSA8903]	Musa, John D.; Faults, Failures, and a Metrics Revolution; IEEE Software, 3/89.
[MUSA87]	Musa, Iannino, Okumoto; Software Reliability: Measurement, Prediction, Application; McGraw-Hill, New York; 1987.
[MUSA8902]	Musa, John D.; Tools for Measuring Software Reliability; IEEE Spectrum, 2/89.
[MYER8611]	Myers, W.; Can Software for the Strategic Defense Initiative Ever be Error-Free?; Computer, 11/86.
[NELS87]	Nelson & Carroll; Tutorial: Fault-Tolerant Computing; IEEE Computer Society Press; 1987.
[NEUG8510]	Neugent, Gilligan, Hoffman & Ruthberg; Technology Assessment: Methods for Measuring the Level of Computer Security; NSB Special Publication 500-133, 10/85.
[NEUM8609]	Neumann, Peter Gabriel; On Hierarchical Design of Computer Systems for Critical Applications; IEEE Transactions, 9/86.
[NYBE89]	Nyberg, Karl; Ada: Sources & Resources 1989 Edition.
[OSTR8806]	Ostrand & Balcer; The Category-Partition Method for Specifying & Generating Functional Tests; Communications of the ACM, 6/88.
[PERK8703]	Perkins & Gorzela; Experience Using an Automated Metrics Framework to Improve the Quality of Ada Software; Proceedings from 5th National Joint Ada Conference, 3/87.
[RAMA8808]	Ramamurthy & Melton; A Synthesis of Software Science Measures and the Cyclomatic Number; IEEE Transactions, 8/88.

[RICH8808]	Rich & Waters; Automatic Programming Myths and Prospects; Computer, 8/88.
[ROLA8606]	Roland, Jon; Software Metrics; Computer Language, 6/86.
[ROMB8703]	Rombach, H. Dieter; A Controlled Experiment on the Impact of Software Structure on Maintainability; IEEE Transactions, 3/87.
[RADC8205]	Rome Air Development Center; Software Testing Measures; RADC-TR-82-135, 5/82.
[RADC8308]	Rome Air Development Center; A Guidebook for Software Reliability Assessment; RADC-TR-83-176, 8/83.
[RADC8403]	Rome Air Development Center; Software Test Handbook - Software Test Guidebook; RADC-TR-84-53, Vol. II; 3/84.
[RADC8502]	Rome Air Development Center; Specification of Software Quality Attributes; RADC-TR-85-37, Vol. I, II, III; 2/85.
[RADC878A]	Rome Air Development Center; Methodology for Software Reliability Prediction and Assessment; RADC-TR-87-171, Vol. I; 8/87.
[RADC878B]	Rome Air Development Center; Software Reliability Prediction and Estimation Guidebook; RADC-TR-87-171, Vol. II; 8/87.
[SCHN8904]	Schneidewind, Norman; Distributed System Software Design Paradigm with Application to Computer Networks; IEEE Transactions, 4/89.
[SCHN8703]	Schneidewind, Norman; The State of Software Maintenance; IEEE Transactions, 3/87.
[SCHN7704]	Schneidewind, Norman; The Use of Simulation in the Evaluation of Software; Computer, 4/77.
[SCHU8805]	Schultz, Herman P.; Software Management Metrics; ESD-TR-88-001, 5/88.
[SELB8703]	Selby, Richard W.; Incorporating Metrics into a Software Environment; Proceedings from 5th National Joint Ada Conference, 3/87.
[SHAT8808]	Shatz, Sol; Towards Complexity Metrics for Ada Tasking; IEEE Transactions, 8/88.
[SHUM88]	Shumskas, Anthony; Why Higher Reliability Software Should Result from Reduced Test and Increased Evaluation; ITEA Journal, 1988.
[SIEG8807]	Siegrist, Kyle; Reliability of Systems with Markov Transfer of Control; IEEE Transactions, 7/88.
[SING87]	Singh, Madan; Systems & Control Encyclopedia - Theory, Technology, Applications; Vol. 3, 6, 8; Pergamon Press; 1987.

[SLON8703] Slonaker, Smith, Prizant & Giles; Development of Multi-Tasking Software in Ada - a Case Study; Proceedings from 5th National Joint Ada Conference, 3/87.

[SPAR8808] SPARTA, TBE & TASC Deliverable to SDIO; Task Order 5 BM/C3 Architectures - (Tools) Subtask 1 Architectural Representation Methodology; Contract No. SDIO84-88-C-0018, 8/15/88.

[SRIV8801] Srivastava & Farr; The Use of Software Reliability Models in the Analysis of Operational Software System; Proceedings from the Institute of Environmental Sciences, 1/13/88.

[STAN85] Stankovic, John A.; Reliable Distributed System Software; IEEE Computer Society Press; 1985.

[STEL8807] Stelovsky & Sugaya; A System for Specification and Rapid Prototyping of Application Command Languages; IEEE Transactions, 7/88.

[TAKA8901] Takahashi & Kamayachi; An Empirical Study of a Model for Program Error Prediction; IEEE Transactions, 1/89.

[TBE8609] Teledyne Brown Engineering Deliverable to ISEC; Generic Tools Requirements for ISEC Ada Transition; MJ86-85-C0244, Contract No. F49642-85-C0244, 9/30/86.

[TBE8703] Teledyne Brown Engineering Deliverable to AMCCOM; Extending McCabe's Cyclomatic Complexity Metric for Analysis of Ada Software; MC87-McCabe II-0003, Contract No. DAAA21-85-D-0010, 3/87.

[TBE8704] Teledyne Brown Engineering Deliverable to AMCCOM; Modified A-Level Software Design Specification for the Ada Complexity Analysis Tool Which Automates the Extended McCabe's Cyclomatic Metric; MC87-McCabe II-0005; Contract # DAAA21-85-D-0010, 4/87.

[TBE8710] Teledyne Brown Engineering Deliverable to CECOM; Software Methodology Catalog; MC87-COMM/ADP-0036, Contract No. DAAB07-86-D-R001, 10/87.

[TBE8808A] Teledyne Brown Engineering Deliverable to SDIO; BM/C3 Architecture Tools, Tool Set Recommendation - TAGS and SADMT and Software Test Plan for the SADMT-Based IORL Simulation Compiler; Contract No. SDIO84-88-C-0018, 8/19/88.

[TBE8808B] Teledyne Brown Engineering Deliverable to SDIO; Technical Report for the Development of a IORL Software Translator to SADMT; Contract No. SDIO-84-88-C-0018, 8/26/88.

[TBE8811] Teledyne Brown Engineering Deliverable to CECOM; Implementation Guidelines for Software Management & Quality Indicators; Contract No. DAAB07-86-D-R001, Draft-9/88, Revised-11/88.

[TENN8809] Tenny, Ted; Program Readability: Procedures Versus Comments; IEEE Transactions, 9/88.

[TITA8809] Titan Systems Deliverable to SDIO; Preliminary Software Test and Evaluation Assessment and Recommendation; 9/30/88.

[TOHM8903] Tohma, Tokunaga, Nagase & Murata; Structural Approach to the Estimation of the Number of Residual Software Faults Based on the Hyper-Geometric Distribution; IEEE Transactions, 3/89.

[TRAC88] Tracz, Will; Tutorial: Software Reuse: Emerging Technology; IEEE Computer Society Press; 1988.

[WEID8602] Weiderman, Nelson; Evaluation of Ada Environments; Report # SEI-86-MR-2, 2/5/86.

[WEIS8810] Weiss & Weyuker; An Extended Domain-Based Model of Software Reliability; IEEE Transactions, 10/88.

[WEYU8809] Weyuker, Elaine; Evaluating Software Complexity Measures; IEEE Transactions, 9/88.

[WEYU8806] Weyuker, Elaine; The Evaluation of Program-Based Software Test Data Adequacy Criteria; Communications of the ACM, 6/88.

[WING8902] Wing, Jeanette & Mark Nixon; Extending Ina Jo with Temporal Logic; IEEE Trans. on Softw. Engineering; 02/89.

[WOLF8903] Wolf, Clarke & Wileden; The AdaPIC Tool Set: Supporting Interface Control and Analysis Throughout the Software Development Process; IEEE Transactions, 3/89.

[WOOD79] Woodward, Hennell & Hedley; A Measure of Control Flow Complexity in Program Text; IEEE Transactions, 1979.

[WU8703] Wu, Basili & Reed; A Structure Coverage Tool for Ada Software Systems; Proceedings from 5th National Joint Ada Conference, 3/87.

[YAU8808] Yau, Nicholl, Tsai, Liu; An Integrated Life-Cycle Model for Software Maintenance; IEEE Transactions, 8/88.

[YAU8011] Yau & Collofello; Some Stability Measures for Software Maintenance; IEEE Transactions, 11/80.

[YAU8606] Yau & Tsai; A Survey of Software Design Techniques; IEEE Transactions, 6/86.

[YU8809] Yu, Shen, Dunsmore; An Analysis of Several Software Defect Models; IEEE Transactions, 9/88.

REF.2 CATEGORY INDICES

METRICS PROGRAMS

1. NUSC - Navy Submarine Combat System Applied Software Metrics Program - 3/15/89
2. IBM - Space Shuttle Programs - 7/29/88
3 USAF R&M 2000 Program - 1/1/89
4. NSWC - Statistical Modeling & Estimation of Reliability Functions for Software (SMERFS) - Dahlgren, VA, 12/88

SDS POLICIES AND PROCEDURES

1. Strategic Defense System Software Policy. Draft 7/19/88

TECHNOLOGY FOR ASSESSMENT OF SOFTWARE QUALITY (TASQ)

1. Database Reports
2 AMC S/W Task Force Brief - 12/14/88
3. POC Cross Reference Draft Report to AMCCOM; Contract No. DAAA21-88-D-0012, Technical Report MC89-TASQ-0003, 1/27/89

SOFTWARE ENGINEERING INSTITUTE (SEI) (received from)

1. The Role of Measurement in Software Engineering; David Card; Proceedings 2nd IEE/BCS Software Engineering Conf., 7/88 [CARD8807]
2. Resolving the Software Science Anomaly; Card & Agresti; Journal of Systems & Software, 1987 [CARD87]
3. Measuring Software Design Complexity; Card & Agresti; Journal of Systems & Software, 1988 [CARD88]
4. Managing Software Maintenance Cost & Quality; Card, Cotnoir, Goorevich; Proceedings IEEE Conference on Software Maintenance, 9/87 [CARD8709]
5. An Empirical Study of Software Design Practices; Card, Church, Agresti; IEEE Transactions; 2/86 [CARD8602]
6. Criteria for Software Modularization; Card, Page, McGarry; Proceedings IEEE Conf. on Software Engineering; 8/85 [CARD8508]

INSTITUTE FOR DEFENSE ANALYSES (IDA)

1. SDS Software Testing & Evaluation: A Review of the State-of-the-Art in Software Testing and Evaluation; IDA Deliverable to SDIO, Report # P-2132, 12/88 [IDA8812]
2. Strategic Defense Initiative Architecture Dataflow Modeling Technique; IDA Deliverable to SDIO, Report P-2035, 4/88 [IDA8804A]
3. A Simple Example of an SADMT Architecture Specification; IDA Deliverable to SDIO, Report P-2036, 4/88 [IDA8804B]

MIL-STDS - IEEE/ANSI/ACM/IDA/Directives/Guidelines/etc.

1. AFSCP-800-14 S/W Quality Indicators 1/20/87
2. AFSCP-800-43 S/W Management Indicators 1/31/86
3. IEEE Std. for Software Productivity Metrics 9/1/88
4. IEEE P1061 Std. for Software Quality Metrics, Draft 2/23/87 & 11/17/88
5. IEEE P982.1/D3.0 Std, A Standard Dictionary of Measures to Produce Reliable Software, Draft 5/88
6. IEEE P982.2/D6, Draft Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software, 5/88
7. IEEE P1044/D3 Draft Standard of a Standard Classification for Software Errors, Faults & Failures, 12/87
8. Orlando II, Panel V, Management Indicators and Quality Metrics, 1/30/87
9. AR 1000-1 Department of the Army, Basic Policies for Systems Acquisition, 5/1/81
10. Draft DoDD 5000.29, Management of Computer Resources in Major Defense Systems, 1/15/86
11. DoDD 3405.1, Computer Programming Language Policy, 4/2/87
12. DoDD 3405.2, Use of Ada in Weapon Systems, 3/30/87
13. Draft DoDD 3405.xx, Computer Software Policy, 10/29/85
14. MIL-STD-2167A, Defense System Software Development, 2/29/88

GEORGE MASON UNIVERSITY

1. Tutorial paper: A User-Oriented Listing of Multiple Criteria Decision Methods; Despontin, Moscarola, Spronk; Belgian Journal of Statistics

2. Metrics & Trade-Off Analysis in Software Design, Production, & Evaluation; Ambrose Goicoechea, GMU [GOIC89RB]

3. A Hierarchical-Multilevel Approach to the Development & Ranking of Systems in Tanning Industry; Goicoechea, Hansen & Henrickson, Computing & Industrial Engineering, 1981 [GOIC81]

4. Evaluation & Ranking of Alternate Systems with Ariadne: An Application to the Economy Car Selection Problem; Goicoechea, Burden, Sanchez; GMU [GOIC89RA]

5. Software Metrics: Their Measurement and Use in Models; Ambrose Goicoechea, GMU [GOIC89RC]

PORTLAND STATE UNIVERSITY

1. Tool Info. (PC-Metric - Developer's Toolbox)

2. Software Science and Weyuker's Fifth Property, Warren Harrison, PSU [HARR89RC]

3. A Note on the Berry-Meekings Style Metric; Warren Harrison & Curtis Cook; Communications of the ACM, 2/86 [HARR8602]

4. MAE: A Syntactic Metric Analysis Environment; Warren Harrison, Journal of Systems and Software, 1988 [HARR88]

5. Applying McCabe's Complexity Measure to Multiple-Exit Programs; Harrison, PSU [HARR89RA]

6. Using Software Metrics to Allocate Testing Resources; Harrison; Journal of Management Information Systems, Spring 88 [HARR8804]

7. Applying Software Complexity Metrics to Program Maintenance; Harrison, Magel, Kluczny, DeKock; Computer Magazine, 9/82 [HARR8209]

8. A Complexity Measure Based on Nesting Level; Harrison & Magel; Univ. of Mo. [HARR89RB]

9. Are Deeply Nested Conditionals Less Readable?; Harrison & Cook; Journal of Systems & Software, 1986 [HARR86]

10. A Micro/Macro Measure of Software Complexity; Harrison & Cook; Journal of Systems & Software, 1987 [HARR87]

TOOLS, COURSES AND NEWSLETTERS

1. Automated Systems Department (TBE-HSV) S/W Tools
2. TBE Accomplishments & Initiatives in Productivity, Training & Automated IV&V Tools April 1988
3. IV&V Methodology & Tools Update - Computer Applications Directorate - April 1988
4. Timing & Sizing Simulation Evaluation - DP-Sim & Network II.5 - Terry Morris - April 1988
5. TBE 1988 BAMD V&V Tools Matrix
6. Logiscope Descriptions
7. McCabe & Associate Tools
8. AT&T Tools for Measuring Software Reliability
9. Dynamics Research Corporation -- AdaMat & Ada Quality Quarterly Newsletter
10. EVB, Software Engineering courses for 1989
11. Ada Information Clearinghouse newsletters

CONFERENCE INFORMATION

1. Proceedings of the Joint Ada Conference - Fifth National Conference on Ada Technology and Washington Ada Symposium; March 16-19, 1987
2. Agenda, NSIA Software Quality and Productivity; March 10-12, 1987
3. Agenda, 9th Annual Conference on Multiple-Criteria Decision Making; August 1990
4. Agenda; Ada Project Management Course by Don Firesmith; Spring 1989.